SRI International

Final Report • Volume 2 • February 1994

CONGESTION AVOIDANCE TESTBED EXPERIMENTS: FINAL TECHNICAL REPORT

ITAD-8600-FR-94-005

Volume 2

Project 8600

Barbara A. Denny, Computer Scientist
Diane S. Lee, Senior Research Engineer
Paul E. McKenney, Sr. Computer Scientist
Danny Lee, Research Engineer
Telecommunications Theory and Technology Program

Prepared for:

National Aeronautics and Space Administration
Ames Research Center
Moffett Field, California 94035

Attn: Dr. Henry Lum, Code RI, M/S: 244-7

and

Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, Virginia 22203-1714

Attn: Major Michael St. Johns

Final Report • Volume 2 • February 1994

CONGESTION AVOIDANCE TESTBED EXPERIMENTS: FINAL TECHNICAL REPORT

ITAD-8600-FR-94-005

Volume 2

Project 8600

Barbara A. Denny, Computer Scientist
Diane S. Lee, Senior Research Engineer
Paul E. McKenney, Sr. Computer Scientist
Danny Lee, Research Engineer
Telecommunications Theory and Technology Program

Prepared for:

National Aeronautics and Space Administration
Ames Research Center
Moffett Field, California 94035

Attn: Dr. Henry Lum, Code RI, M/S: 244-7

and

Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, Virginia 22203-1714

Attn: Major Michael St. Johns

Approved by:

Michael S. Frankel, Vice President and Director
Information, Telecommunications, and Automation Division

SRI International 333 Ravenswood Avenue • Menlo Park, CA 94025-3493 • (415) 326-6200 • FAX: (415) 326-5512 • Telex: 334486

CONTENTS

FIGURES AND TABLES

FIGURES

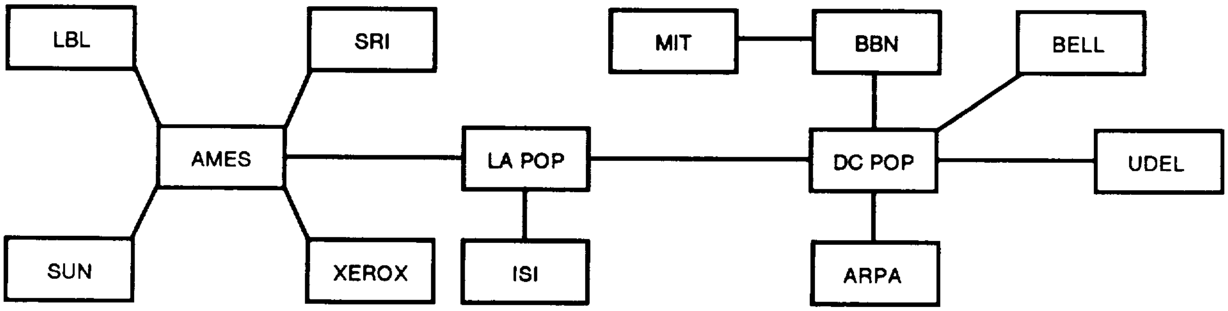
TABLES

1 INTRODUCTION

DARTnet is the Advanced Research Projects Agency (ARPA) Research Testbed Network whose purpose is to provide an experimental platform for network research in traffic control, resource management, routing algorithms, and advanced networking applications. It is a cross-country T1* network that connects research sites via T1 tail circuits. The switches are Sun Microsystems, Inc. (Sun) SPARCstations,† running+ a 4.1.1 Sun-based kernel in order to provide an open development platform for investigating new technology. The switches use Sun's High Speed Serial Interface (HSI/S) card to transmit and receive packets into the network. Figure 1 illustrates the current DARTnet topology.

This document is part two of the final technical report describing the work performed by SRI International (SRI) under SRI Project 8600. Part two, comprising volumes 2 and 3, covers SRI's work on DARTnet in the area of DARTnet in the area of Congestion Avoidance Testbed Experiments (CATE). Our goals in this effort were to advance the state of the art in benchmarking networking performance and traffic control by developing support tools for network experimentation, by designing benchmarks that allow various algorithms to be meaningfully compared, and by investigating new queueing techniques that better satisfy the needs of best-effort and reserved resource traffic.

This document is organized as follows. The first volume (Volume 2 of the final technical report) describes the work performed and the results obtained. It begins with an overview of the project, followed by sections describing each task in detail (Sections 3 and 4 deal with our benchmarking work; Sections 5 and 6 deal with our work in traffic control), and ends with our concluding comments and a list of references, followed by appendices.



LBL:	Lawrence Berkeley Laboratory	LA POP:	Los Angeles Point of Presence
MIT:	Massachusetts Institute of Technology	DC POP:	Washington, DC Point of Presence
BBN:	Bolt Beranek and Newman Inc., Massachusetts	UDEL:	University of Delaware
BELL:	Bellcore, New Jersey	SUN:	Sun Microsystems, Inc., California
AMES:	National Aeronautics and Space Administration Ames Research Center, California	XEROX:	Xerox Palo Alto Research Center, California
		ISI:	Information Sciences Institute, University of Southern California (USC)

Figure 1. DARTnet Topology

*Due to hardware constraints, the network currently operates at 1.344 Mb/s. instead of 1.536 Mb/s.

†All product names mentioned in this document are the trademarks of their respective holders.

The appendices, including all reports produced for each task (as Appendices A, B, D, E, and F), are as follows.

- Appendix A: *Traffic Generator Software Release Notes* [McKenney, Lee, and Denny 1993].
- Appendix B: *Experiment Design for CATE* [McKenney and Denny 1993].
- Appendix C: Baseline Measurements.
- Appendix D: *A Report on Stochastic Fairness Queueing (SFQ) Experiments* [Denny 1993a].
- Appendix E: *A Hybrid Algorithm for Combining Best-Effort and Resource Reservation Service* [Denny 1993b].
- Appendix F: *Annotated Bibliography for Congestion Control and Resource Reservation* [Lee and Denny 1993].

The second volume of this report (Volume 3 of the final technical report) contains the source code of all software developed on this project. All of this software is available via anonymous FTP on ftp.erg.sri.com.

2 OVERVIEW

2.1 OBJECTIVES

DARTnet provides an excellent environment for executing networking experiments. Since the network is private and spans the continental United States, it gives researchers a great opportunity to test network behavior under controlled conditions. However, this opportunity is not available very often, and therefore a support environment for such testing is lacking. To help remedy this situation, part of SRI's effort in this project was devoted to advancing the state of the art in the techniques used for benchmarking network performance.

The second objective of SRI's effort in this project was to advance networking technology in the area of traffic control, and to test our ideas on DARTnet, using the tools we developed to improve benchmarking networks. Networks are becoming more common and are being used by more and more people. The applications, such as multimedia conferencing and distributed simulations, are also placing greater demand on the resources the networks provide. Hence, new mechanisms for traffic control must be created to enable their networks to serve the needs of their users. SRI's objective, therefore, was to investigate a new queueing and scheduling approach that will help to meet the needs of a large, diverse user population in a "fair" way.

2.2 APPROACH

To improve current benchmarking capabilities, SRI developed a general-purpose traffic generation/networking loading tool called Traffic Generator (TG), and a set of associated analysis tools to reduce and comprehend the data collected by the TG. The TG enables researchers to create high-quality and repeatable experiments on packet-switched networks. The TG is driven by a control language (script) that specifies various operating modes, protocols, addressing functions, traffic parameters, and execution times, so that researchers know exactly what traffic is being created and when. SRI did not, however, want to recreate existing tools. Therefore, SRI decided to use a public-domain graphing tool called grtool to display the results collected by the TG, such as delay and throughput. Simple perl scripts were written that take the data from the TG and place it in a tabular file that grtool can use. A series of experiments for testing various algorithms' resource reservation and allocation capabilities were also developed. By executing these experiments, researchers can quantitatively evaluate and compare the abilities of different algorithms to meet the resource needs of its users. These benchmarks are not dependent on a specific network topology and therefore may be applicable to a wide range of users; however, DARTnet is a suitable environment for performing these experiments.

A queue is a natural place to control the traffic flow. Historically, network service has been provided through a first-in, first out (FIFO) queueing discipline. This simple form of traffic control has proved to be inadequate under increasing service demands, due to its lack of fair service to individual users and its inability to support the sophisticated resource demands of new applications. Under previous ARPA sponsorship, SRI developed a probabilistic variant of fair queueing known as Stochastic Fairness Queueing (SFQ). It was designed to give fair service to best-effort traffic, i.e. traffic with no delivery constraints, and to scale well as the number of users increases. Our approach, therefore, is to investigate SFQ in DARTnet by implementing and experimenting with it and then combine it with a resource reservation algorithm to satisfy the demands of users who require a certain level of service from the network.

2.3 ACCOMPLISHMENTS

SRI fulfilled the goals stated above. In particular, SRI

- Developed, and released to the public, support tools consisting of the TG and data reduction tools based upon the output of the TG. This traffic generator has attracted users at many other locations, including commercial, government and academic sites.
- Designed a series of experiments for testing the resource reservation and allocation capabilities of various algorithms.
- Implemented and experimented with Stochastic Fairness Queueing (SFQ). Our experiments show that SFQ provides fair utilization of available resources, prevents starvation, and degrades gracefully under overload conditions.
- Implemented and experimented with a hybrid algorithm combining SFQ with VirtualClock. This algorithm can satisfy the needs of both best-effort and reserved-resource traffic. BBN plans to include it in an upcoming release of ST-II.
- Prepared an annotated bibliography of recent articles relating to congestion-control and resource-reservation techniques in high-speed networks [Lee and Denny 1993].

3 EXPERIMENT SUPPORT

In this section, we provide an overview of our work relating to providing a software infrastructure to support experiments on DARTnet. The main effort of this task was devoted to developing a traffic generator that would allow a researcher to perform repeatable experiments. After surveying existing tools, we noted the lack of suitable software for performing network experimentation in a quantitative fashion. Researchers would use existing applications, such as FTP and ping, to verify the functionality of a new algorithm; but there was no software to help them evaluate the quantitative behavior in a consistent and repeatable manner. The Swedish Institute of Computer Science Protocol Implementation Measurement System (SPIMS) was available, but it has some shortcomings. In particular, the statistics are very high-level, such cumulative counts of number of packets sent and received. The granularity of the information collected, therefore, is not fine enough to determine the kind of behavior network researchers would be interested in, such as delay characteristics and loss patterns. SRI, therefore, developed a general purpose traffic generator and some associated tools for analyzing the log files produced by the traffic generator. These tools are discussed in more detail below.

3.1 TRAFFIC GENERATOR

The TG is a tool for creating high-quality and repeatable experiments on packet-switched networks; it can also be used to characterize the performance of packet-switched network communication protocols. It executes as a source and sink program that enables experimenters to generate one-way traffic streams and gather statistical data about the transmission and reception of each stream. The TG program is controlled by a simple but flexible specification language that allows access to different operating modes, protocols, addressing functions, experiment traffic parameters, and execution times. At present, packet length and packet offer rate can be specified according to the following distributions: constant, uniform, exponential, and 2-state Markov. Each Markov state is associated with its own distribution, thereby allowing the nesting of different distributions within the 2-state Markov. The TG relies on the Network Time Protocol (NTP) for underlying synchronization.

Traffic sent and received by the TG is recorded in binary log files. Each log file contains the time of an event, the event type, and the event data that is dependent on the event type. For receive and transmit events, a datagram/segment identification number is printed. A filter program, dcat, is provided with the TG; dcat converts binary TG output to an ASCII text file.

In the current version, packet traffic can be sent via the TCP, UDP, and ST-II protocols. As new transport and network-layer protocols are developed, the TG can easily be enhanced to support them. Applications that want to use TG can also be easily integrated by adding the appropriate calls to the datagram or stream protocol structure.

The software was developed in C under SunOS 4.1.1 and thus should be easily portable to other UNIX operating systems. It has been built for SGI machines and machines running Solaris 2.1 with minimum effort (without ST-II support since this is not present in those systems). The software is also in the public domain. The users include

- The DARTnet community (SRI, Xerox PARC, BBN, LBL, MITRE, the University of Massachusetts [UMass], UDEL, and ISI)
- Academic institutions (USC and the University of Maryland [UMD]-NASA Center for the Commercial Development of Space)

- Commercial companies (3M Minnesota, Bell Northern Research [BNR] Canada, Bellcore, Sequent, Wellfleet, and AT&T Research)
- Government facilities (CECOM; Rome Laboratory [Network Design Facility]; and NRL).

The software release notes, which constitute a user's manual for the TG, can be found in Appendix A. The source code for the TG and dcat is in Volume II, Section 2 of this report and is available for anonymous FTP on ftp.erg.sri.com in the pub/tg directory.

3.2 ANALYSIS TOOLS

The ASCII files produced by the TG can be processed using other tools, such as perl, awk, and grtool, to help interpret the information collected. To support our work involving baseline measurements, SFQ, and the hybrid algorithm, we have written perl scripts that compute summary statistics for the ST-II and UDP protocols. The statistics include average offer rate, average throughput, average delay, and delay variance. The scripts also can produce tables that can be used by grtool, a public-domain graphing tool, to graphically represent the data regarding throughput, delay, and packet loss. These scripts have also been released to other members of the DARTnet community to aid in the evaluation of their work.

The perl scripts can be found in Volume II, Section 3, and are available via anonymous FTP on ftp.erg.sri.com in the pub/tg directory.

4 BENCHMARKS

In this section, we present an overview of our benchmarking work on DARTnet. Benchmarking is important because it provides a means for evaluating the performance and behavior of whatever is under test, whether it is the entire network or a specific algorithm or protocol. Furthermore, scant information is available on the performance of newly developed networking technologies and algorithms, because the preliminary efforts in development focus on demonstrating and testing what a protocol or algorithm does, rather than how well it performs. There is a need, therefore, to define a set of experiments that evaluate the performance of these new protocols and algorithms and allow meaningful comparisons to be made.

4.1 CONGESTION AVOIDANCE TESTBED EXPERIMENTS

To satisfy this need, SRI has written a document that serves as an experimenter's manual by defining a set of experiments that evaluate the resource allocation and reservation capabilities of three different kinds of algorithms: best effort, type of service, and resource reservation. This document, called *Experiment Design for CATE,** [McKenney and Denny 1993] is provided as Appendix B to this report. The experiments are divided into three classes, according to the kind of algorithm involved. Best-effort experiments assume all users have equal rights to network resources, including traffic such as mail, where there are no hard requirements for delivery. Type-of-service experiments assume that users have different rights to network resources; these rights are expressed in terms of priorities, delay constraints, and throughput limits. These experiments, therefore, test the ability of the algorithm to honor these type-of-service constraints. Resource-reservation experiments assume explicit throughput reservation and evaluate the enforcement of a reservation.

Each experiment is defined in terms of a general framework that includes the objective, procedure, measurements taken, and desired results. The procedure is described in high-level terms so that the document can be applied to different networking environments (for example, DARTnet is an adequate environment for executing the experiments). Researchers can use these experiments to quantitatively evaluate and compare the ability of an algorithm to satisfy the service demands placed upon today's networks. Tables 1, 2, and 3 present an overview of the experiments defined within the CATE document. These tables include each experiment's objective and the metrics used to evaluate performance.

4.2 BASELINE MEASUREMENTS

As preliminaries to the more elaborate tests outlined in the CATE document, we conducted simple baseline tests to characterize the packet-switching performance of DARTnet. All tests utilized UDP because we wished to measure the performance of the network, not the transport protocols.

The first test was designed to verify the operation of the network by providing an indication of the throughput and transmission loss as a function of packet generation rate and packet size. The tests were of a parameterized nature, varying the packet generation rates, packet size, and number

*CATE: Congestion Avoidance Testbed Experiments.

Table 1. Best-Effort Experiments

EXPERIMENT	CPU IDLE TME	ETE DELAY	ETE DELAY VARIANCE	ETE PACKET LOSS	ETE THROUGHPUT	OFFER RATE	OFFER LOAD	FAIRNESS	PATH UTILIZATION	CONVERSATION-SETUP DELAY	BLOCKING PROBABILITY
Determining Overhead	•	•		•		•	•				
Overload Behavior with Cooperative Sources	•	•	•	•	•		•	•	•		
Overload Behavior with Uncooperative Sources	•	•	•	•	•		•	•	•		
Delay for Low-Bandwidth User	•	•	•	•	•		•	•	•		

Table 2. Type-of-Service Experiments

EXPERIMENT	CPU IDLE TME	ETE DELAY	ETE DELAY VARIANCE	ETE PACKET LOSS	ETE THROUGHPUT	OFFER RATE	OFFER LOAD	FAIRNESS	PATH UTILIZATION	CONVERSATION-SETUP DELAY	BLOCKING PROBABILITY
Honoring Priorities	•	•	•	•	•		•	•	•		
Honoring Delay Constraints	•	•	•	•	•		•	•	•		
Honoring Throughput with Cooperative Sources	•	•	•	•	•		•	•	•		
Honoring Throughput with Uncooperative Sources	•	•	•	•	•		•	•	•		

Table 3. Resource-Reservation Experiments

EXPERIMENT	CPU IDLE TME	ETE DELAY	ETE DELAY VARIANCE	ETE PACKET LOSS	ETE THROUGHPUT	OFFER RATE	OFFER LOAD	FAIRNESS	PATH UTILIZATION	CONVERSATION-SETUP DELAY	BLOCKING PROBABILITY
Honoring Existing Reservations	•	•	•	•	•		•		•	•	
Blocking Correlation	•	•	•	•	•		•		•	•	•
Honoring Future Commitments	•	•	•	•	•		•		•	•	

of T1 hops for a single traffic stream. A TG script sent 1000 UDP packets at a constant rate and size. Sizes were 32, 64,128, 256, 512, 768, 1024, 1280, and 1500 bytes. Rates were 1, 2, 10, 20, 50, 100, 200, 500 and 1000 packets/s. For packet generation rates greater than 50 packets/s, a delay of 10 s was inserted before the beginning of each new action. The most notable problem we experienced was unexpected packet loss, even at loads below the capacity of the network (see Appendix C, Subsection C.4, for an example).

We then designed a simple two-stream test whose total capacity was ~1.26 Mb/s, to test the behavior of the network with traffic in opposing directions utilizing one cross-country path and a cut-through path (see Figure 2). The cross-country stream from SRI to UDEL used an exponential distribution with a mean interarrival rate of 0.01 (100 packets/s) and a mean packet size of 576 bytes. The opposing exponential stream ran from ISI to SRI with a mean interarrival rate of 0.01 and a mean packet size of 1000 bytes. This test again pointed to large "blackouts" in the network. Appendix C, Subsection C.4, contains graphs illustrating the blackouts in the network.

During some of our early experiments, we experienced out-of-order arrivals (see Appendix C, Subsection C.4, for an example). This phenomenon would seem to be impossible, due to the linear topology of the network, and we could find no explanation for it; however, it has not recurred for quite some time. We note it here as an interesting anomaly.

Due to the fragility of the network at the end of 1991, we decided to pursue simple measurements of network performance. To this end, we defined a test methodology for determining throughput. The throughput of the network is defined to be the maximum point where

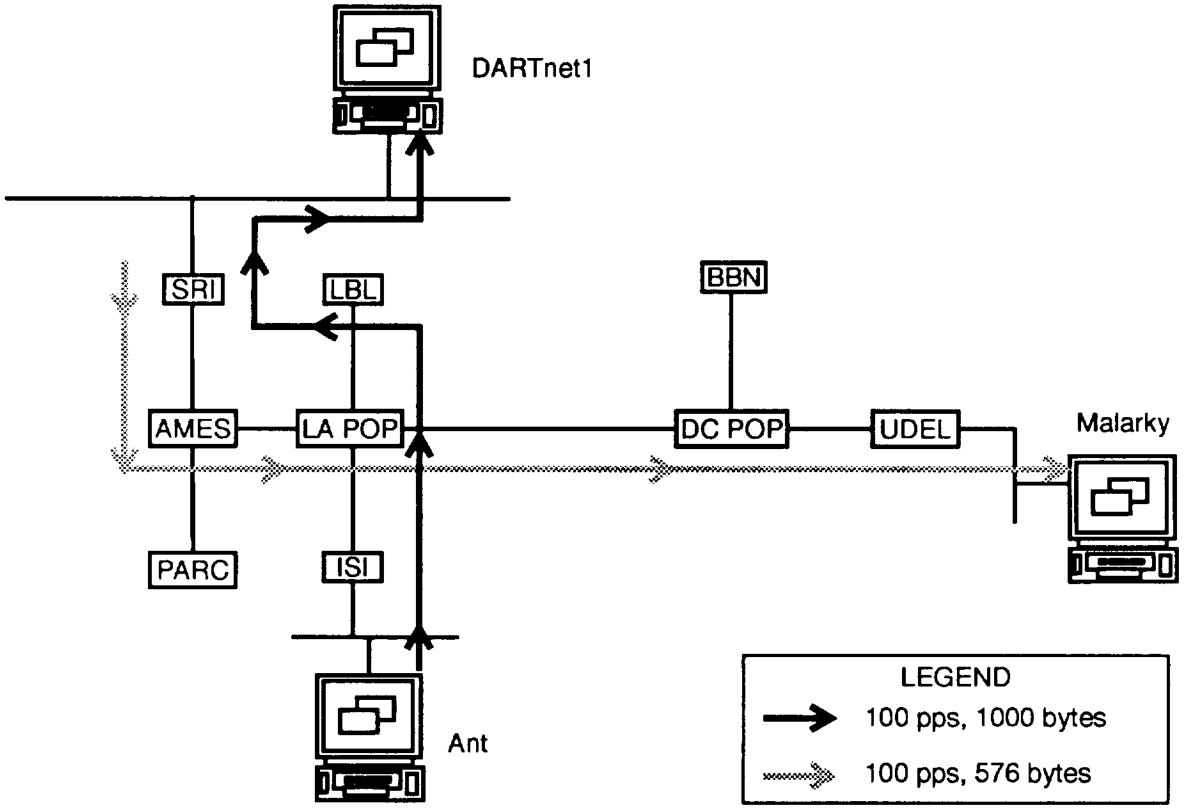


Figure 2. Two-Stream Test Traffic Flow

delay is approximately constant and there is no packet loss (ideally). We then chose a technique to measure throughput. We decided to fix the packet length and vary the offer rate until we found the "knee" of the curve. The point below the beginning of the knee is the throughput for that packet size. Figure 3 shows the end-to-end delay for this experiment and illustrates the states during the experiment and the point at which the throughput is established. Note that each interval marked on the X axis indicates a new offer rate.

We then executed this methodology on DARTnet several times during the lifetime of the project, using a constant distribution for interarrival rate and packet size and paths of two different lengths. (See Figure 4.) This repetition was necessary because of changes in the configuration of the network: modifications were made to the kernels to support new software and to fix bugs discovered during the tests; the line speed was reduced as a work-around to solve a hardware design problem in the interface board*; and service providers were changed. Figure 5 shows the throughput obtained at 1.536 Mb/s and 1.344 Mb/s when Performance Systems, Inc. (PSI) was the service provider, and at 1.344 Mb/s when the service provider was changed to Sprint.† Appendix C, Subsections C.1, C.2, and C.3, contain the graphs obtained from these experiments. Each run was executed at least twice, but only one run is recorded in the appendix, unless the runs yielded markedly different results.

For these values, it is interesting to note that throughput increased as a function of packet size, although we expected throughput to be relatively constant for the given offer rate. It has been speculated that the increase might result from "bunching" of the packets at the source, due to the granularity of the UNIX clock's interaction with the packet-scheduling algorithm of the traffic generator. This theory has not been verified. Neither the length of the path, however, nor the line speed, seem to have affected the results significantly. At most, the length of the path varied the

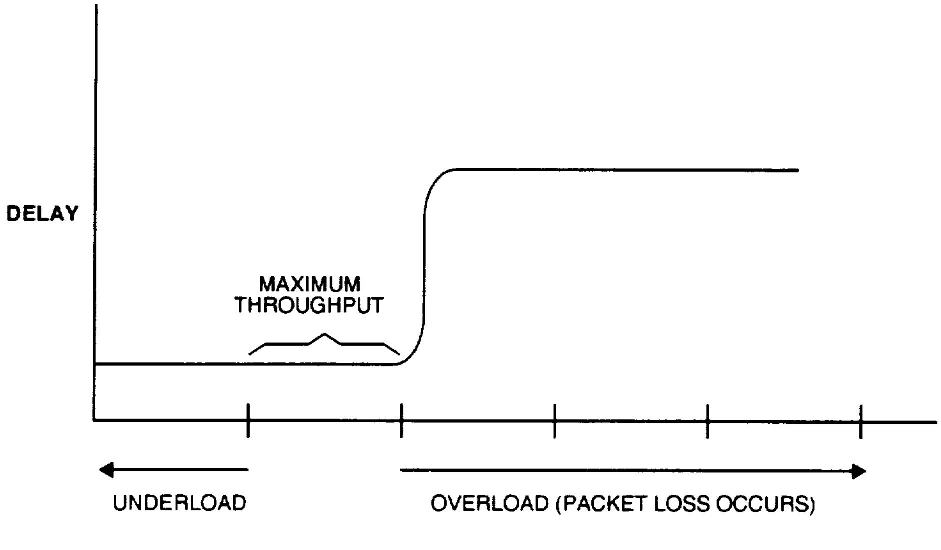


Figure 3. Determining Throughput

*The line speed was changed to 1.344 Mb/s to help reduce the possibility of overruns by providing more time to move the data from the FIFO.

†We were unable to complete the throughput measurements for Sprint, due to lack of funding, so we have included only those packet sizes tested.

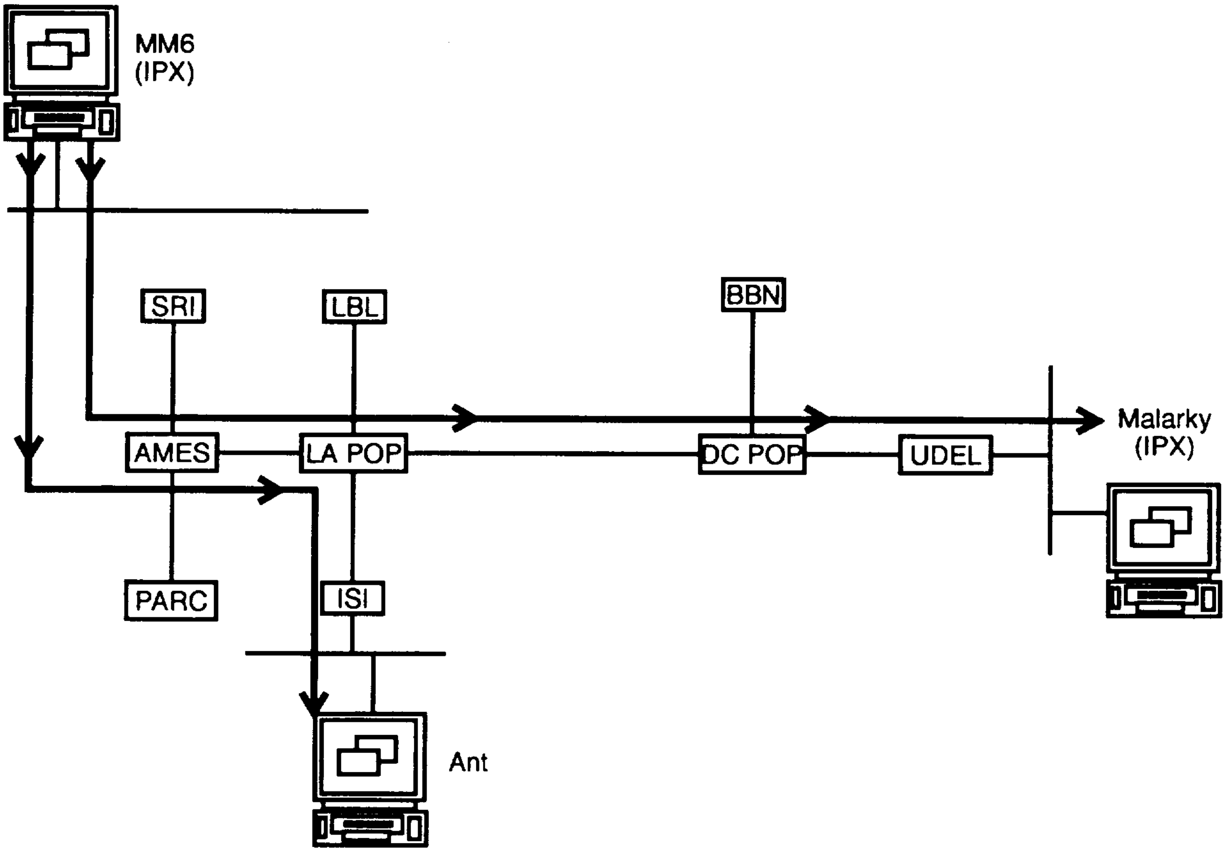


Figure 4. Throughput Traffic Flow

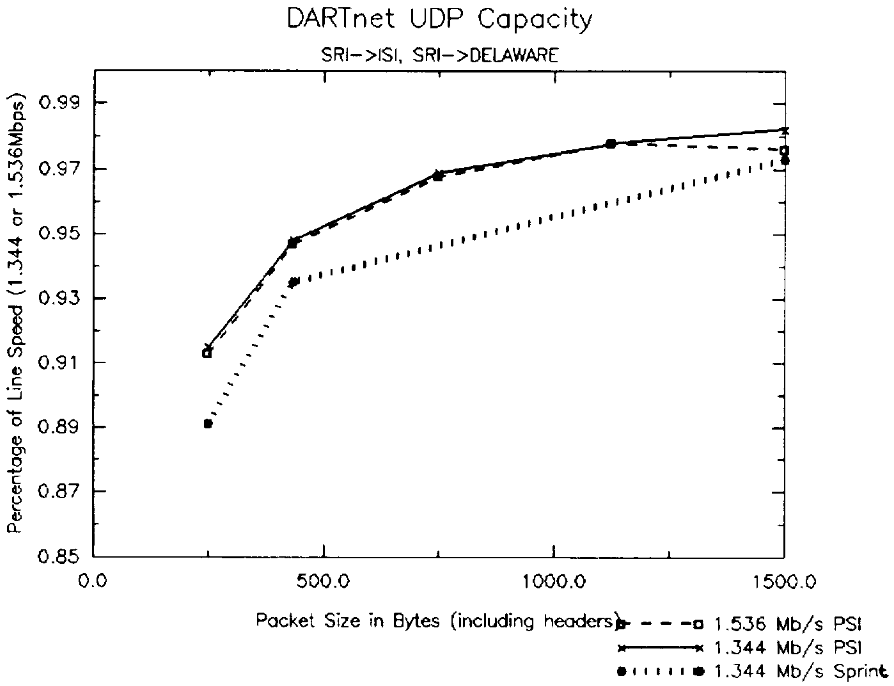


Figure 5. DARTnet Throughput

packet/s count by only plus or minus one packet. This variation did not occur frequently. The lack of perturbation to the throughput results due to line speed change is evident on examination of the change in line speed from 1.536 Mb/s to 1.344 Mb/s. The reduction in throughput is proportional to the change in line speed for the same provider.* The throughput reduction obtained when the service provider changed has not been explained. Originally, we believed that the reduction might be due to the addition of nodes to the network; but upon examining the software in the driver, we concluded that this explanation was unlikely. No further work could be done, because of the exhaustion of funds.

4.3 MEASUREMENT RESULTS

Our experiments revealed many problems with the network, as indicated above. Most of the severe network blackouts, however, were caused by inadequate hardware. Many subtle problems were uncovered that related to the design of the HSI board. Since then, the network has been engineered to support testing in overload conditions by adding additional cards to the switch, so there is one T1 card per line. The software driver has also been carefully modified to overcome inadequacies in the design of the board. At the time of this final report, no known problems remain; but the network has not been adequately tested or characterized for the current software and topological configuration.

*The increase in throughput for 1500-byte packets was not investigated. It is assumed to be the result of a bug fix.

5 STOCHASTIC FAIRNESS QUEUEING

In this section, we present an overview of the work we performed on DARTnet relating to traffic control. Traffic control is necessary, because current datagram networks are vulnerable to congestive collapse when the offered load approaches or exceeds their capacity. Although several end-to-end congestion-avoidance algorithms have been proposed, none of them has been shown to perform optimally in high-speed, high-bandwidth delay product networks. This has led some researchers to believe that source control of data flow is not enough and that the gateways must also participate in congestion avoidance. To this end, it has been proposed that the gateways use a fairness queuing algorithm to insulate users from the activities of other (possibly ill-behaved) users. Fairness queuing, as originally proposed by Demers, Shenker, and Keshav [1989], is inefficient because it requires each conversation to be mapped to a separate queue. Stochastic Fairness Queuing, as developed by Paul McKenney [McKenney 1991] provides an efficient means of dispersing traffic on a fixed number of queues and is suitable for high-speed networks.

5.1 OVERVIEW

SFQ is a packet-by-packet fairness queueing algorithm that uses a hashing function to provide a simple mapping from source/destination address pair to a queue. The mapping is not guaranteed to be unique, and the choice of hash function is critical to good performance. The hash functions we have chosen attempt to maximize randomness in order to ensure different hash values for most address pairs. In addition, the hashing seed is perturbed periodically to reduce recurring collisions, thereby decreasing the chance that streams will fail to receive their "fair share" of the available resources because they always share the same queue. The queues are serviced in round-robin fashion.

The advantages of SFQ are that it is conceptually easy and fairly simple to implement. SFQ is more efficient in space requirements and needs fewer memory references to find a queue than strict fair queueing. Its simplicity may also make SFQ a prime candidate for hardware/firmware implementation and thus a suitable candidate for high-speed networks. Furthermore, it would be easy to combine it with other algorithms for improved performance. However, SFQ has its disadvantages (1) for efficiency, the number of fixed queues must be large; (2) perturbation of the hash seed must be done carefully, to ensure the ordered delivery of packets; and (3) SFQ provides only a local means of traffic control.

5.2 IMPLEMENTATION

We completed an implementation of SFQ for the SunOS 4.1.1-based DARTnet kernel. Although this implementation is based on SunOS, it should port easily to other platforms. Our implementation consists of an array of finite-length queues and a doubly linked "active" list for packet transmission, which includes only queues that are not empty (see Figure 6). The existing UNIX queuing macros are replaced with a small set of macros and subroutines, which implement the SFQ queuing discipline with the data structures outlined above. We also implemented several nonrotating and rotating hash functions that use a seed and the source and destination IP address to determine the queue. The nonrotating hash function may perform better; it is the one currently used. Our first implementation relied on a simple mask to perform the final mapping from the

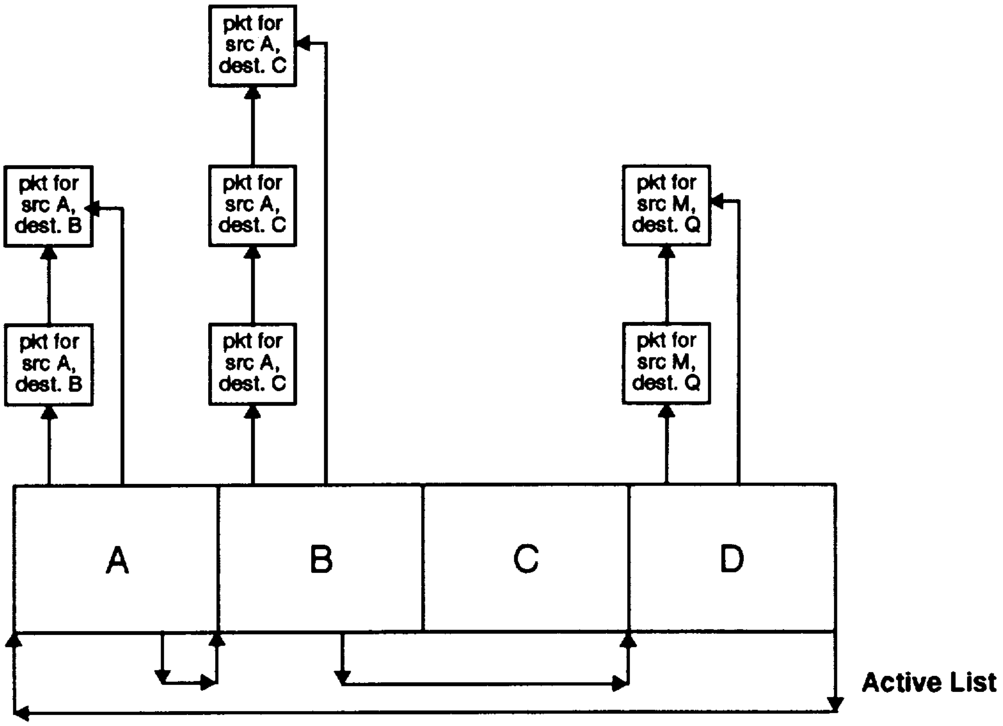


Figure 6. SFQ Data Structure

results of the hash function to the bucket index. This method resulted in too many collisions, and perturbing the hash seed did not change the behavior. We therefore changed the mapping from the hash function to the queue size, so that it uses the mod operator. A software implementation that works for an operand whose value is a (power of 2) plus 1 was written; the C-language operator was not used because the SPARC implementation was considered too slow. Other improvements are possible but have not been implemented. For example, the technique of dropping a packet when a queue is full may be replaced by buffer stealing. In addition, the criterion for perturbation may be changed. Currently, the seed is changed when all queues to an interface are empty, to prevent reordering problems. In reality, this situation may never arise at a "busy" gateway; therefore, a different criterion for perturbation should be investigated. The source code for SFQ is provided in Volume II, Section 4.

5.3 EXPERIMENTATION

To verify SFQ's resource control mechanism, we performed various experiments on DARTnet. These experiments were designed to show

- Fair utilization of available resources
- Prevention of starvation for streams whose demands are small
- Graceful degradation under overload conditions (as streams are added, bandwidth used by each stream decreases fairly)
- Resource usage compared to FIFO queueing.

A report covering the experiments [Denny 1993a] is included in Appendix D. In general, the experiments do show that SFQ is better than FIFO queueing at allocating bandwidth equally among a set of flows. SFQ also prevents a stream from dominating the available bandwidth, which seems to occur in FIFO queueing (i.e., if a stream demands more than its share of the available bandwidth, with FIFO queueing that stream receives a disproportionate amount, compared to streams that demand less than their share). Furthermore, SFQ seems to reward "nice" users of the network by providing a lower variance in delay and more throughput when their resource demand is less than their available share. Both SFQ and FIFO queueing seem to degrade fairly well as the network becomes saturated, and to recover well as the network becomes less congested. Not unexpectedly, FIFO queueing is a little more efficient than SFQ—FIFO delays are shorter and the throughput slightly higher because SFQ requires more processing. However, the performance difference between the two queueing disciplines is relatively small.

However, the experiments do point out some interesting behavior. FIFO queueing can behave better than SFQ with seed perturbation. We recommend further evaluation of the hash function and the seed perturbation technique: their current selection probably contains weaknesses that cause this unexpected behavior. SFQ also seems to possess good scaling properties. To verify this, more experiments with a larger number of streams from more hosts should be executed and examined, including the staggered introduction of streams. Staggering the streams may prove important, because graphs in the degradation experiment revealed some unexpected increases and decreases in throughput, which should be examined. These phenomena may again be due to the interaction of the hash function with seed perturbation, but may also be related to some other unknown problem.

6 HYBRID ALGORITHM FOR BEST EFFORT AND RESOURCE RESERVATION SERVICE

In this section, we present an overview of our enhancements to the general SFQ mechanism discussed in the previous section. In particular, SFQ was designed to support best-effort traffic (traffic that has no hard delivery requirements). However, new applications, such as audio and video, require certain delivery constraints to be met that SFQ was not designed to provide. Many new scheduling algorithms are being developed to meet these resource needs. A hybrid algorithm combining the strengths of SFQ with a resource reservation algorithm would better meet all the needs of the users of a network. The following section describes our new hybrid algorithm, which combines the strengths of SFQ with a prototype resource reservation algorithm, and the results we obtained from implementing and experimenting with this hybrid algorithm on DARTnet. A detailed report [Denny 1993b] covering this work is included in Appendix E.

6.1 DESIGN

VirtualClock [Zhang 1990] was selected as the resource reservation algorithm to be combined with SFQ, because other algorithms and approaches were not mature enough or were not in a form that was available for our use. VirtualClock is a rate-based scheduling algorithm that orders packets in a single queue according to a timestamp. This timestamp is based on the average arrival rate specified in the reservation request. For the routers to receive this resource request, a mechanism must exist to pass the flow specification. ST-II provides this functionality, and since a VirtualClock implementation was included in the release of the ST-II protocol, we are using ST-II.

Since we decided to keep two logical queues, one for SFQ and one for VirtualClock, the hybrid algorithm only requires the definition of the packet-scheduling algorithm that selects the queue from which the next packet will be transmitted. Two different approaches were explored. The first is a simple priority scheme where the VirtualClock queue always has priority over the SFQ queue. The second approach is an attempt to provide "fairer" service. Reserved resource traffic should meet its scheduling requirements; however, this should not be done in a manner that could starve the best-effort traffic, as in the simple priority scheme above. (Such starvation would occur in the case where the reserved resource traffic is high enough that there is always something in the VirtualClock queue.) The second approach, therefore, tries to interleave the packets in both queues while still guaranteeing the reserved resource traffic its average rate. To achieve this, we reinterpreted the timestamp calculated by VirtualClock for packet ordering as the time to send this packet as well. The scheduling algorithm, then, is as follows:

- If the VirtualClock queue is not empty, check the packet timestamp at the head of the VirtualClock queue.
    - If the packet timestamp is later than now and the SFQ queue is not empty, take the packet from SFQ; otherwise, take the packet from the VirtualClock Queue.
    - If the packet timestamp equals now or is earlier than now, remove the packet from the VirtualClock Queue.
- If the VirtualClock queue is empty, try to remove the packet from SFQ.

Note that this algorithm always tries to find a packet to send if one is available. An improvement to the algorithm could be a method for determining how far in the future packets should be sent from the VirtualClock queue before a best-effort packet is taken. This improvement would prevent any reserved resource traffic from being "late" because of a best-effort packet.

6.2 IMPLEMENTATION

The prototype implementation of this algorithm was done in version 13 of the DARTnet kernel, using release 1.12 of BBN's implementation of ST-II and VirtualClock. The code supplied by BBN also included their traffic control interface abstraction, so we decided to try out their model in our implementation of the algorithm [Lynn 1993]. This interface consists of a series of routines that the algorithm designer must supply. This consisted of defining routines for initialization, classification, enforcement, enqueueing, and dequeueing. We therefore defined a new interface that included a joining of the calls provided in our previous implementation of SFQ and BBN's implementation of VirtualClock. Of course, the dequeueing routine included the algorithm described above, with subsequent calls to the appropriate queueing structure.

The main difficulty in our implementation involved the design of the dual queue. Much of the code written within the kernel assumes that the pointer to the head of the queue is in the ifnet structure. Since we had two different queues, this was not possible. However, as part of the design of their traffic control abstraction, BBN extended the ifnet structure to include some space for pointers that are algorithm specific. We used one of these pointers to hold the SFQ head pointer. Any code that involved accessing the queue pointer was then carefully modified to take into account this dual queue design. This modification also required changes to the HSIS driver and the routine if_down in if.c.

6.3 EXPERIMENTATION

To evaluate the performance of our hybrid algorithm, we choose to use the default algorithm in the traffic control interface that BBN supplied.* This baseline algorithm consisted of a single queue where VirtualClock and FIFO techniques were merged. Established streams in ST-II used VirtualClock, while best-effort traffic used FIFO (any FIFO packets were always at the end of the Virtual Clock queue). The other two algorithms under test consisted of the versions of the hybrid algorithm described above.

The experiment was designed to demonstrate the two key design goals of our hybrid algorithm. In particular, these goals are that

- Reserved resource traffic streams acquire their requested average packet-generation rate
- Best-effort traffic streams receive the benefits of SFQ (streams receive equal portions of the available bandwidth, and hosts that send less traffic than their allocated bandwidth are rewarded).

*We originally planned to use a traffic control algorithm that only provided FIFO queueing and no resource management for our baseline measurements. However, we experienced problems setting up ST-II streams with TG for this algorithm. The streams could not be established, so we could not document this algorithm in this report. This appears to be an ST-II implementation issue in the kernel, i.e., what the proper response should be if no resource guarantees are ever possible.

In general, the results of the experiment we performed do show that one can effectively combine SFQ with VirtualClock to achieve the benefit of SFQ (equal access to the available bandwidth), while preserving the average rate requested by the reserved resource traffic. Interleaving the two queues based on time did result in slightly better throughput than using a simple priority scheme. We expect the interleaving algorithm to do at least as well as or better than the simple priority version, under almost all conditions. Improvements to the interleaving process are also possible. It may be better to compute a time into the future that should be used to choose the queue from which the next packet is to be transmitted. This would reduce the probability of the "late" delivery of a VirtualClock packet that inadvertently gets behind a best-effort packet.

7 CONCLUSION

In this report, we have presented our work on DARTnet. DARTnet is a valuable testbed that allows researchers to gain experience and insight into network behavior for both existing and new protocols and algorithms. Performing baseline measurements and experiments with our new queueing mechanisms gave us some useful insights and we learned various lessons in the execution of our tasks. We experienced at first hand the difficulties of fielding a new network. Experienced network implementors using mature networking technology still must perform extensive experiments to verify correct operation before they can begin the more complicated task of evaluating protocols and algorithms, and understanding network dynamics. We also need better tools to undertake this effort: better instrumentation in the switches, a better distributed model of experiment control and traffic generation, automated collection of instrumented data, improved post-analysis tools, and real-time visualization aids that display selected network information. Furthermore, net synchronization aids in the analysis of the data; and the traffic generator relies on Network Time Protocol (NTP) to synchronize clocks to the millisecond. Synchronization to one ms currently takes about a day. However, synchronization to a "few" milliseconds can be achieved in about an hour. In a test environment, convergence time is important, because of the necessity of switching kernels for different research efforts.

The current state of DARTnet, however, is unknown, because the kernel used in DARTnet has changed since our measurements and testing with the new service provider was never completed. These tests should be completed, to verify that no problems persist in the network. Once this is done, researchers will have more confidence in their evaluation and testing of any new protocols or algorithms.

Traffic control is a rich and important field of study that we are only beginning to explore. We have shown that SFQ is superior to FIFO queueing in its distribution of available resources. SFQ also can be implemented efficiently and can be effectively combined with other queueing techniques. In particular, combining SFQ with VirtualClock can effectively satisfy the demands of resource reservation traffic while providing best-effort traffic with equal access to the remaining bandwidth. Interleaving the packets from the VirtualClock queue and SFQ seems to be better than a simple priority scheme. Interleaving provides fairer access to the network by allowing best-effort traffic to be sent as soon as possible, without interfering with the scheduling requirements of the reserved resource traffic, and also achieves slightly better throughput than a simple priority scheme.

However, more work is required. The current seed perturbation technique for SFQ is not satisfactory because network congestion may result in "unfair" behavior: New techniques need to be designed and tested. SFQ was designed to scale well; however, this property was never really studied or tested and should be investigated before this technique is deployed in a large networking environment. Our hybrid algorithm, which combines SFQ with VirtualClock, could probably be improved by calculating a window during which the next virtual packet should be sent, so as to prevent "late" delivery of the packet. A better mechanism for handling bursts of traffic should also be developed. Currently, any packet that causes a stream with a resource reservation to exceed its allocated limit during its averaging interval will be dropped. This method is inefficient, because enough bandwidth may be available to handle the packet. Other approaches should be developed and tested. For example, one could place this packet in the SFQ queue instead, although the result may be out-of-order delivery. If this delivery order is important, it may be better to add some kind of priority scheme to packets in the Virtual Clock queue, so that any packet over its limit is tagged; then, if it becomes necessary to drop a packet, a tagged packet is first chosen. Another issue that should be examined is the following question: How does this type of approach fit in with a hierarchical allocation policy where an agency wants to ensure its allocated share of the network capacity? Finally, we need to experiment with other traffic control algorithms to compare their performance with our hybrid algorithm and to understand their behavior in general. The CATE document [McKenney and Denny 1993] provides a good beginning for these experiments and should be used to this end.

REFERENCES

Clark, D., S. Shenker, and L. Zhang. 1992. "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," *Proc. of ACM SIGCOM*, pp. 14-26.

Demers, A., S. Keshav, and S. Shenker. 1989. "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. of ACM SIGCOMM*, pp. 1-12.

Denny, B.A. 1993a. *A Report on Stochastic Fairness Queueing (SFQ) Experiments*, ITAD-8600-TR-93-62, SRI International, Menlo Park, California (March).

Denny, B.A. 1993b. *A Hybrid Algorithm for Combining Best-Effort and Resource-Reservation Service,* ITAD-8600-TR-93-168R, SRI International, Menlo Park, California (October).

Floyd, S. 1992. "A Report on Link-Sharing Experiments," draft paper, Lawrence Berkeley Laboratory, Berkeley, California.

Jacobson, V., and S. Floyd. 1993. *Hierarchical Resource Management*, unpublished work, Lawrence Berkeley Laboratory, Berkeley, California.

Lee, D.S., and B.A. Denny. 1993. *Annotated Bibliography for Congestion Control and Resource Reservation*, ITAD-8600-TR-93-74, SRI International, Menlo Park, California (March).

Lynn, C. 1993. "Net Interface Extension Memo," draft technical note, Bolt Beranek and Newman Inc., Cambridge, Massachusetts.

McKenney, P.E. 1991. "Stochastic Fairness Queueing," in *Internetworking: Research and Experience*, Vol. 2, pp. 113-131.

McKenney, P.E., and B.A. Denny. 1993. *Experiment Design for CATE*, ITAD-8600-TR-93-191, SRI International, Menlo Park, California (July).

McKenney, P.E., D.Y. Lee, and B.A. Denny. 1993. *Traffic Generator* Software Release Notes, ITAD-8600-TN-93-28, SRI International, Menlo Park, California (1 February).

Topolcic, C., ed. 1990. "Experimental Internet Stream Protocol, Version 2 (ST-II)," RFC 1190.

Zhang, L. 1989. *A New Architecture for Packet Switching Network Protocols*, doctoral dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Zhang, L. 1990. "VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks," *Proc. of ACM SIGCOMM*, pp.19-29.

Appendix A

TG USER MANUAL

Traffic Generator
Software Release Notes

Paul E. McKenney
Danny Y. Lee
Barbara A. Denny
Information and Telecommunication Sciences Center
SRI International
Menlo Park, CA

February 1, 1993

1 Introduction

This document describes a packet Traffic Generator (TG) program that can be used to characterize the performance of packet-switched network communication protocols. The TG program generates and receives one-way packet traffic streams transmitted from the UNIX[1] user process level between traffic source and traffic sink nodes in a network. Different protocols, or versions of the same protocol, may be tested to ascertain differences in performance.

The TG program is controlled by a simple, but flexible, specification language that allows access to different operating modes, protocols, addressing functions, and experiment traffic parameters. The specification language allows traffic of several different packet lengths and interarrival time distributions to be generated. In the current implementation, the TCP and UDP transport protocols and raw ST-II protocol are supported.

While the name "Traffic Generator" focuses on the packet generator aspect of the program, the TG program also serves as a traffic sink. In order to record packet transmission statistics, a TG configured as a traffic sink must be active to receive the test traffic. The received traffic stream is logged in a file by the traffic sink for post-test analysis. When connection-oriented transport services are used, a traffic sink is needed to accept an incoming connection request. The TG serving as the traffic source always logs datagram transmit times. This mode of operation may be useful for analyzing network blocking characteristics or for loading a network.

Depending upon whether connection-oriented or connectionless transport services are used, one or more traffic sinks may be needed. For datagram protocols like UDP or ST-II, multiple traffic sources may send their traffic to the same sink process (a many-to-one relationship). The TG serving as a traffic sink logs all received datagrams.

When TCP protocols are used, there must be a one-to-one mapping between a traffic source and a traffic sink for each connection. Each invocation of the TG program is able to sustain a single stream; the TG is currently unable to service more than one active connection at a time. Consequently, if two connections are desired between a pair of nodes, two traffic sinks are required.

2 User Guide

This section describes how to use the TG program. We describe the function of the TG program and the syntax of its command specification language.

Note that keywords and names of programs are shown in `typewriter` font, while parameters (values that you supply) are shown in *italic* font. Optional parameters are enclosed in

[1] UNIX is a registered trademark of UNIX System Laboratories, Inc.

brackets [ ]: these parameters may be safely omitted. During our discussion of the syntax of TG specification and log files, angled brackets < > are used to delimit metavariables.

In our discussion, all numeric parameters can be entered as decimal, octal, hexadecimal integer, or floating-point numbers. In general, a number may be an integer (1), a decimal (1.0), or a floating-point number (1E-3). Octal (010) and hexadecimal (0x10) numbers are also supported, although they are not too useful for our application.

All time values supplied to the TG program are expressed as hours:minutes:seconds (H:M:S) separated only by colons and without intervening white spaces. These time fields are not limited, so you can if you desire work only in seconds or minutes.

If the H:M:S time format is not followed, the fundamental unit of measure for time defaults to units of "seconds," or fractions thereof. Note that the unit of measure for the packet size random variables is bytes.

Internally, the TG deals with most parameters as double precision floating-point values, so that even very small values are represented correctly. The precision of a number will be preserved, in spite of the potential inability of a system to generate packets at the specified resolution.

2.1 Software Distribution Hierarchy

The software distribution hierarchy consists of several top-level directories, which follow common UNIX conventions. Two programs form the TG distribution: the first is the traffic generator (`tg`), and the second is a filter (`dcat`) for reading TG log files.

The directories and their contents are as follows.

- `./bin` – Executable versions of SunOS[2] 4.1.2 binaries
- `./docs` – Documents and other notes
- `./examples` – Sample TG specification files[3]
- `./src` – Source code for the TG program.

2.2 Building the TG Program

The TG is released with source code so that enhancements can be added to support new protocols or additional statistical distributions that may be needed for future experimentation. Note that the TG distribution includes compiled SunOS 4.1.2 binaries in the `./bin` directory.

[2] SunOS is a trademark of Sun Microsystems, Inc.

[3] TG specification files are also referred to as scripts.

To rebuild the traffic generator, change to the `./src` directory and execute the **make** command. Your system will also need the **lex** and **yacc** programs to rebuild the TG program. Note that the software has been compiled and tested only on the SunOS 4.1.2 operating system; however, the software should work on other Berkeley UNIX-based operating systems.

2.3 Invoking the TG

The syntax for invoking the TG program is as follows:

```
tg [-f] [ -i input-file ] [ -o output-file ]
```

By default, the TG program reads the standard input (stdin) device for a TG specification file and writes its log file to the standard output device (stdout). The traffic generator accepts the `-i` and `-o` options, which override the default input/output behavior. The `-i` option specifies an input file that contains TG test specification entries. The destination of the log file can be changed with the `-o` option. The `-f` option specifies that the buffer be flushed to the log file after every write.

Alternatively, specification and log files may be stored on disk by using the I/O redirection features present in many command interpreter shells. For example, the following command in the UNIX **csh** redirects the contents of *input-file* into the traffic generator program, and writes the log file to *output-file*.

```
tg < input-file > output-file
```

2.4 TG Specification File

A specification file consists of entries in the following forms, and in the order specified:

< *start-time* >
< *association-spec* >
< *tg-action-list* >

The *start-time* clause indicates the time that a test run is supposed to start, relative to the time the clause is parsed. Upon satisfying the *start-time* clause, the actions listed in the *association-spec* and *tg-action-list* clauses are executed. Before test traffic can be generated, an *association-spec* entry must be provided to specify an association, or binding, between a traffic source and a traffic sink. The *association-spec* also provides the ancillary test parameters used in a test session. The *tg-action-list* is a list of *tg-action* clauses; each clause specifies an action to be performed, such as causing the TG program to generate test traffic with specified distributions.

A file inclusion capability is supported, which allows the TG specification to be contained in multiple files. A line of the form

```
include "filename"
```

causes the named file to be read and parsed. Double quotes must enclose the filename. The maximum depth of inclusion is precompiled into the program and can be changed by redefining `MAX_INCLUDE_LEVEL`.

2.4.1 Starting Time

The experiment starting time is specified relative to the current time as an offset value. The offset value is in modulus-time form, where the supplied parameter specifies the next time boundary that an action will take place. The syntax for the start time is

```
on H:M:S
```

During an invocation of the TG, the **on** clause can appear at most once in a specification file.

This scheme is relatively independent of local time. The advantage of the scheme is that it allow scripts to be rerun without modification. It also allows scripts to synchronize to within a few milliseconds of each other, despite having been started several seconds out of synchronization (e.g., due to network delay to geographically remote hosts, or "telephone slew" between human operators). The quality of the synchronization is limited only by the ability of the time protocol (e.g., Network Time Protocol) to synchronize time, and by operating-system factors such as scheduling granularity and delays.

For example:

```
on 15
```

starts at the next fifteen-second interval (`x:00, x:15, x:30, or x:45`), while

```
on 1:00
```

starts at the beginning of the next minute.

For the very patient:

```
on 1:00:00
```

starts at the beginning of the next hour.

After the **on** clause has been satisfied, the *association-spec* and *tg-action-list* in the following lines are executed. Note that the **on** clause is designed to provide session-level synchronization between the TG processes, and does not cause traffic to be generated. Traffic is not generated until a *tg-action* clause has been parsed, as discussed in Section 2.4.3.

2.4.2 Association Specification

An *association-spec* is a binding between a traffic source and a traffic sink. An *association-spec* may take one of the following two forms:

protocol local-address **server** [*quality-of-service*]

or

protocol remote-address [*quality-of-service*]

Traffic Source/Sink Modes. The TG program can serve both as a traffic source and a traffic sink. The word **server** in the first form of the *association-spec* clause is a keyword that indicates that the traffic generator should act as a traffic sink. Once in the **server** mode, the TG program will wait indefinitely for incoming connections or traffic.

In the traffic sink mode, the TG acts upon the received messages as appropriate. In some cases, response traffic is returned when the sink is in the `interactive` mode. When a response is required, the server responds with a packet of the length specified in the *tg-action-list* entry.

When the **server** keyword is used, the *local-address* field is used to specify the local address at which the TG will accept traffic. The address must be for the local host, as the TG binds to the named port. Note that the IP address may be specified as `0.0.0.0`.

Protocols. The *protocol* argument is a string (such as `tcp`, `udp` or `straw`) that selects the appropriate transport-layer protocol to be used to carry the test traffic. The current implementation supports only TCP, UDP and raw ST-II protocols; however, the internal protocol table may easily be expanded to accommodate new protocols as they are developed.

Addressing. The current implementation does not resolve host names. All addresses must be supplied in the standard Internet dotted address form (e.g., 128.18.4.100.1234). Note that the port number is appended to the address. For example, the address `128.18.4.100.1234` specifies a rendezvous point of port `1234` at the host with the address `128.18.4.100`.

Traffic streams are multiplexed by protocol port addresses. In many UNIX implementations, port numbers less than 1024 are privileged and can only be accessed by programs with root access privileges. It is recommended that port numbers between 1024 and $2^{16} - 1$ be used. If a port is already being used, select a different port number and reexecute the program.

The *remote-address* field selects the destination endpoint of the association; e.g., the destination of a datagram or a peer in a connection. (Support for one-to-many ST-II streams is not yet available in TG.) Note that addresses used in the association specification include the port number. The example below sets up the TG program so that subsequent transmissions are sent to the process running on 128.18.6.100 and listening at port 2345.

```
udp 128.18.4.29.2345
```

Quality of Service. The `quality-of-service` (QOS) field consists of the following entries, specified in any order:

```
average bandwidth number
peak bandwidth number
average delay number
peak delay number
average loss number
peak loss number
interval number
mtu number
rcvwin number
sndwin number
```

These options were designed to serve as subscription/connection setup time profiles. In the current TCP and UDP protocol implementation, these QOS options may be omitted. Eventually, these options may be used or ignored as the protocol's connection setup function sees fit. Currently only **`rcvwin`** and **`sndwin`** are supported under the TCP protocol. These fields control the number of bytes in the receive and send window. All other options are ignored, though the numbers are parsed and inserted in the protocol data structure.

The ST-II protocol uses some of the QOS options to specify the resources that should be reserved for the flow. The **`mtu`** option must be specified and is interpreted as the average packet size in bytes (even though the name implies otherwise). The **`interval`** option must be specified as seconds between packet arrivals and is used to compute the average packet rate. The minimal required bandwidth in number of bytes per second may be specified by either the **`peak bandwidth`** option; or (if it is not specified) the **`average bandwidth`** option; or (if it is not specified) the product of the packet size and rate from the **`mtu`** and **`interval`** options. Note that these options are used to specify the reservation, not the actual traffic profile.

2.4.3 TG Action List

Traffic is generated via *tg-action-list* clauses. The *tg-action-list* is a list of *tg-action* elements, each of which consists of

[`at` *time-literal*] *tg-action*

The optional `at` clause specifies a time relative to the start time that the associated *tg-action* will execute. The *time-literal* is a time specified in the H:M:S colon format. If the `at` clause is omitted, the action will commence upon completion of the previous action.

A *tg-action* consists of either

setup

by itself or

wait [*time-literal*]

by itself or

arrival *distribution*
length *distribution* [responselength *distribution* [patience *time-literal*]]

in that order. These entries may be followed by a list of any or all of the following, specified in any order:

data *number*
packet *number*
seed *number*
time *time-literal*

The **setup** clause forces an association setup to occur. If no **setup** clause is present, an implicit setup will occur at the time specified by the **on** start-time clause. It is illegal to have more than one setup clause; if a setup clause is present, it must precede all *tg-action* clauses that generate traffic.

The **wait** clause causes the traffic generator to pause, as specified by the time-literal argument, or by the succeeding **at** clause (or forever, if there is no time-literal argument and this is the last *tg-action*: this is useful for servers). The **arrival** clause specifies the interarrival time distribution, and the **length** clause specifies the packet length distribution. If the **responselength** clause (which may be abbreviated **resplen**) is present, then each packet sent will contain an integer selected from the corresponding distribution and encoded suitably for stream protocols. If, in addition, the **patience** clause is present, the traffic generator will expect to receive a packet of the specified length within the specified time, aborting if its patience is ever exhausted. The **data** clause limits the total amount of data to be sent in bytes (mimicking, for example, a file transfer operation). Similarly, the **packet** clause specifies the number of packets to send. The **seed** clause sets the random-number generator seed (before generating random variables for the arrival and length clauses); while the **time** clause limits the total amount of time spent transmitting.

Where possible, the program attempts to deduce implicit time clauses and **at** clauses from those of surrounding *tg-action-list* clauses: in the following example, deduction is not possible:

```
arrival exponential 0.030 length 120
data 1000000
arrival exponential 0.060 length 240
```

The program cannot know in advance how long it will take to send one million bytes of data through the network.

2.4.4 Statistical Distributions

Statistical distributions can be associated with packet interarrival and packet length random variables. The interarrival distribution is specified with the keyword **arrival**, while the packet length distribution is preceded by the keyword **length**. Currently, four types of distributions are supported—additional distributions are planned and will be added when implemented. Four keywords are used to specify the distribution that can be used with a *tg-action* clause.

constant *value* or *value*
uniform *max* or **uniform** *min max*
exponential *mean* or **exponential** *mean min max*
markov2 *number distribution number distribution*

The **constant** distribution always returns the number specified in the supplied parameter; and if desired, the **constant** keyword may be omitted entirely. The **uniform** distribution requires a number specifying the maximum value of the open interval [0, *max*) from which the random number is to be drawn. If the left endpoint is also specified, a random number is selected from the open interval [*min*, *max*). The **exponential** distribution is the classic distribution with the specified mean. As with the **uniform** distribution, it is possible to restrict the values that are returned from the exponential distribution to only those within the open interval [*min*, *max*); the exponential distribution is unchanged otherwise. Specifying an upper limit for the exponential distribution will force only reasonably sized packets to be issued; otherwise there is a small probability that an infinite size packet may be generated, and which will result in an error being reported. The **markov2** keyword specifies a two-state markov distribution: the first number gives the mean time in state 1; the second number gives the mean time in state 2. Each state is associated with its own distribution; these distributions could themselves be markov2 distributions, if desired. As shown above, the italicized parameter *distribution* is a placeholder for a keyword from the set: **constant, uniform, exponential, and markov2**.

2.5 Examples

The following simple example illustrates the specification entries for a TCP traffic source.

```
on 0:15
tcp 128.18.4.97.2345
at 5 setup
at 6 arrival exponential 0.1 length exponential 576
seed 321423 time 10
```

The `on 0:15` command instructs the TG program to wait until the next 15-second time boundary approaches before initiating any protocol activity. Since the keyword `server` is absent in the *association-spec* line, TG will initiate a TCP traffic connection to the node address 128.18.4.97, port 2345. The `setup` command, which is activated 5 seconds after synchronizing on the closest 15-second boundary, initiates a connection request to the traffic sink node. One second later, traffic will be sent across the connection with an exponential interarrival distribution with a mean of 0.1 seconds. The TCP segment size is also exponentially distributed with a mean of 576 bytes. The last line sets the random number seed and limits the test duration to 10 seconds.

A TG must serve as a traffic sink, before TCP traffic can be exchanged with the traffic source. The specification entries for a traffic sink are simpler, and are as follows:

```
on 0:15 tcp 128.18.4.97.2345 server
at 1.1 wait
```

Note that the `server` keyword is present, and that the TG enters into the TG `wait` state, 1.1 seconds after the start time synchronization boundary.

Other examples of TG scripts can be found in the `./examples` directory supplied with the software distribution.

3 Log File

The TG program records all notable events in a log file for post-test analysis. With the exception of the header descriptor, the TG log file is encoded in binary form to conserve storage space. Some binary fields are encoded with a variable-length code for additional space savings. Such fields are stored as a sequence of bytes: seven bits per byte in little-endian format. The sign bit is set to one, to indicate that the field continues into the next byte, or cleared to zero to indicate that this is the last byte forming a number. Note that the value of the number is unaffected if the last byte is 0x00. This allows a trailing 0x00 to be used to flag the number as being special in some way.

In general, the supplied `dcat` program should be used to expand a TG log file into readable text form. The `dcat` filter is distributed with the release in the `./bin` and `./src/dcat` directories.

3.1 Log File Header

An ASCII header prefaces each log file and provides general information about the test session that generated the log data. The information stored in the header of the log file includes the following:

- Header file version
- Version of the TG program used
- Name of the host that collected the test data
- Start time of the test
- Name of the protocol under test
- Address family indicator
- TG specification file.

The ASCII header portion of the log file may be viewed directly by using the UNIX filter **head** to extract the first several lines of the file. The ASCII header is delimited by the strings <Begin TG Header> and <End TG Header>. Note that the first character on the next new line following the ASCII header contains a binary TG log record.

3.2 Log File Record Format

The `dcat` program converts TG log records into a readable ASCII format and prints its output to stdout. The output from the `dcat` program can be fed into **awk** or other similar string processing programs. `dcat` output lines have the following format:

```
<Event Timestamp> <Event Type> <Address> <Event Data>
```

An example of the output from the `dcat` program is shown below.

```
Time             Type             Address                 Id       Length
---------------------------------------------------------------------------
0.003678         Setup
10.061812        Accept           128.18.6.90.1626        Association 4
11.041100        Transmit         128.18.6.90.2346        0        1460
11.041244        Receive          128.18.6.90.1626        0        1460
16.006682        Teardown
```

Event Timestamp. The event timestamp field is printed as the number of seconds since the start of the test run: that is, the time since the **on** clause was satisfied. Note that the event timestamp is not specified relative to the **at** clause. Although the timestamp field is printed with microsecond resolution, the actual precision of the timestamp depends upon the granularity returned by the `gettimeofday` system call.

Event Types. TG event types include the majority of the common events that can be observed from the UNIX user-process level. The event types include the following:

- Receive—indicates a packet was received.

- Transmit —indicates a packet has been transmitted.

- Setup—indicates protocol initialization is complete.

- Accept—indicates a connection has been established.

- Teardown—indicates a connection has been torn down.

- Error—indicates an error in the program was detected.

Addresses. The address field specifies the TG peer that generated the event. Depending upon the event type, the address may indicate the source of a received packet or the destination of a transmitted packet. Addresses take the form of an Internet address with a port number appended to it.

Event Data. Event data are present only for certain event types. For Receive and Transmit events, a datagram/segment identification number is printed. The length of the received datagram/segment is also printed. For Accept events, the address of the connection peer is printed, together with the association number. For Setup and Teardown events, no other data is provided.

An identification number is used to reference a datagram or a segment within a stream. For stream protocols, the identification field specifies the position of the first received byte in the stream: consequently, the identification field added to the length field gives the total number of bytes transferred at the event time.

An Error event type is also provided to record errors as they occur during the execution of the program. Depending upon when the error takes place, an address may or may not be recorded in the log file. The error codes are described in log.h and are reprinted below.

```
#define LOGERR_INTFMT       1       /* Script format error */
#define LOGERR_MEM          2       /* Out of memory */
#define LOGERR_2SETUP       3       /* Two connections were established */
#define LOGERR_GETTIME      4       /* gettimeofday() failed */
#define LOGERR_SELECT       5       /* select() failed */
#define LOGERR_FCNTL        6       /* fcntl() failed */
#define LOGERR_GETPEER      7       /* getpeername() failed */
```

3.3 Internal Log File Format

For those wishing to write their own filters or data manipulation programs in order to process a log file directly, the log file is organized as a sequence of records, with each record having as a minimum the record type, control, and timestamp fields. The form of the binary log file is similar to the tables printed by the **dcat** program. The source code in the **dcat** filter should provide a good starting point. The format is as follows.

```
< Receive > < Control > < Time > < Address > < Id > < Size > [ Errno ]
< Transmit > < Control > < Time > < Address > < Id > < Size > [ Errno ]
< Setup > < Control> < Time > [ Errno ]
< Teardown > < Control> < Time > [ Errno ]
< Accept > < Control > < Time > <Address > < Association > [ Errno ]
< Error > < Control > < Time > < Address > < Error Type >
```

Consult the **tg** and **dcat** source code for the elaboration of each field. As detailed above, the [**Errno**] field holds the UNIX error number, and will very likely not be present in most records.

4 Concluding Comments

The TG software distribution is available from sparkyfs.erg.sri.com (128.18.4.39) via anonymous FTP. The TG software is provided "as is" without express or implied warranties. SRI International (SRI) will not be held responsible for loss of data or inaccuracies resulting from the use of this program. Although SRI does not support the software, we welcome your comments and bug reports. If you use this software, please register your name and address with SRI by sending electronic mail to "dartnet@erg.sri.com" so that we can notify you of updates and software bugs.

Appendix B

BENCHMARK DOCUMENT

Technical Report • July 1993

EXPERIMENT DESIGN FOR CATE

Paul E. McKenney, Senior Systems Programmer
Barbara A. Denny, Computer Scientist
Information and Telecommunications Sciences Center

SRI Project 8600

ITAD-8600-TR-93-191

Prepared for:

National Aeronautics and Space Administration
Ames Research Center
Moffett Field, California 94035

Attention: Dr. Henry Lum, Code RI, M/S:244-7

and

Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, Virginia 22203-1714

Attention: Dr. Paul Mockapetris

Approved:

Michael S. Frankel, Vice President and Director
Information, Telecommunications, and Automation Division

1 INTRODUCTION

Network performance and behavior are key concerns to network researchers, implementors, and users alike. As we move toward higher- and higher-speed networks, new architectures and protocols are being designed to address the issues inherent in the uses of the technology. Researchers are developing new solutions to the problems created by using these new technologies and the demands placed on the networks by new applications, such as voice and video. Efforts by various researchers, however, still focus on demonstrating and testing what a protocol does, rather than how well it does it. There is a need, therefore, to define a set of experiments that evaluate the performance of these new protocols and algorithms and allow meaningful comparisons to be made.

SRI International (SRI) is pleased to submit this document, which serves as an experimenter's manual by defining a set of experiments that evaluate the resource allocation and reservation capabilities of three different kinds of algorithms: best effort, type of service, and resource reservation. By executing these experiments, researchers can quantitatively evaluate and compare the ability of these algorithms to satisfy the service demands that today's networks need to address. These experiments are known collectively as the *Congestion-Avoidance Testbed Experiments (CATE)*.

A general framework for defining an experiment is presented in Section 2. Briefly, for each experiment the objective, procedure, measurements taken, and desired results are provided. The procedure is described in high-level terms so that the document can be applied to different networking environments. Definitions of the measurements are found in Section 2, along with suggested programs for performing the experiments and obtaining the desired measurements for a UNIX-based* environment.

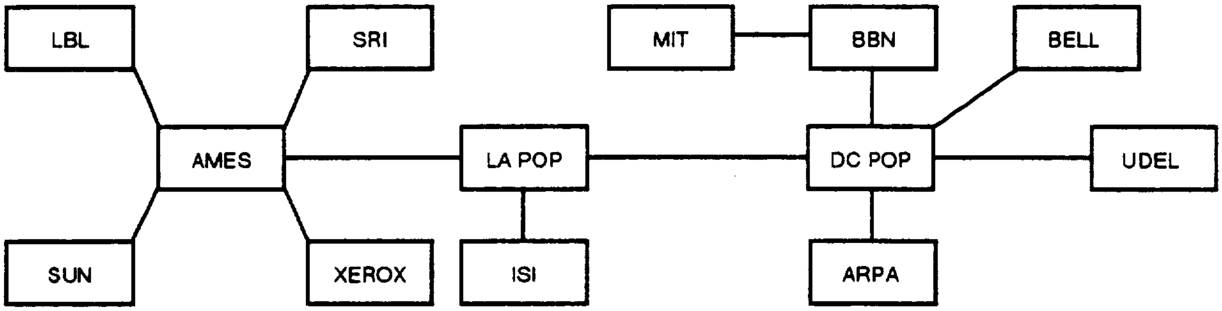
The actual experiments are divided among the next three sections according to the needs the algorithms address. Best-effort algorithms (Section 3) assume that all users have equal rights to network resources, and include applications such as mail, where there are no hard requirements for delivery. Type-of-service algorithms (Section 4) assume that users have different rights to network resources and that these rights are expressed in terms of priorities, delay constraints, and throughput limits. Resource reservation algorithms (Section 5) support explicit throughput reservation with respect to time. The experiments in the different sections can be applied to algorithms designed for different needs, to test the algorithms' ability to meet those requirements as well.

As mentioned previously, CATE can be used in many different networking environments (for example, DARTnet† [described below]) since minimal assumptions are made concerning resources or topology. The only real resource constraint is the experimenter's ability to generate enough traffic to saturate the capacity of the network being used and to have sole access to the network(s) in which the experiments are performed. The necessary topology has been kept as simple as possible, so that only the availability of a single, fixed-path route is assumed. Some experiments, therefore, are not included (for example, the adaptability of the algorithm is not tested).

*All product names mentioned in this document are the trademarks of their respective holders.

†DARTnet: the DARPA (now ARPA) Research Testbed Network.

DARTnet is an experimental platform for network research in traffic control, resource management, routing algorithms, and advanced networking applications. It is a cross-country T1[*] network that connects research sites via T1 tail circuits. Figure 1 illustrates the current DARTnet topology. This network provides sufficient resources to execute CATE.



LBL:	Lawrence Berkeley Laboratory	LA POP:	Los Angeles Point of Presence
MIT:	Massachusetts Institute of Technology	DC POP:	Washington, DC Point of Presence
BBN:	Bolt Beranek and Newman Inc.	UDEL:	University of Delaware
BELL:	Bellcore, New Jersey	SUN:	Sun Microsystems, Inc.
AMES:	National Aeronautics and Space Administration Ames Research Center	XEROX:	Xerox Palo Alto Research Center, California
		ISI:	Information Sciences Institute, University of Southern California

Figure 1. DARTnet Topology

2 EXPERIMENT OVERVIEW

This section provides the general framework that will be used to describe the experiments in the following sections, the elements and techniques common to every experiment, and a summary of the experiments. For each experiment, the objective, procedure, measurements taken, and desired results are provided. To apply this document to a specific network requires an explicit description of the traffic conditions during the time of the experiment, including the source and destination of each stream and traffic characterization. Since no assumptions are made about the network(s) being tested, traffic conditions are not discussed in this report. It is assumed that each experiment will be run at least twice for every algorithm under study. More runs may be necessary if the results of the runs vary too greatly. It is also assumed that each experiment will study the behavior of at least two competing algorithms: one of the algorithms will be considered a baseline to which the other algorithm is compared. For example, First-In First-Out (FIFO) and Stochastic Fairness Queueing (SFQ)[†] may be the two algorithms under study for the best-effort experiments. The remainder of this section presents an overview of the material covered in each experiment and the procedure for gathering the metrics used in a particular experiment.

[*] Due to hardware constraints, the network currently operates at 1.344 Mb/s instead of the standard T1 rate of 1.536 Mb/s.

[†] McKenney, P. 1991. "Stochastic Fairness Queueing," in *Internetworking: Research and Experience*, Vol. 2, pp. 113-131.

2.1 OBJECTIVE

For each experiment, an objective must be stated. This objective will demonstrate a principle relating to the algorithm, such as the ability to honor future reservations.

2.2 PROCEDURE

The traffic flow during the experiment is dependent on the topology and bandwidth of the network under test and the objective of the experiment. Since we are not making assumptions about the network on which these tests are run, no specific scenarios are provided. However, the general conditions and the approach will be described. It is also suggested that each stream exist for at least 240 seconds.

To obtain the measurements described below in Subsection 2.3, it is assumed that some kind of traffic generator will be used. For repeatability of the experiments, we suggest using the SRI-developed Traffic Generator (TG). The TG is a tool used to create high-quality and repeatable experiments on packet-switched networks. It executes as a source and sink program that enables experimenters to generate one-way traffic streams and gather statistical data about the transmission and reception of each stream. The TG is driven by a control language (script) that specifies different operating modes, protocols, addressing functions, traffic parameters, and execution times. At present, packet lengths and packet offer rates can be specified according to the following distributions: constant, uniform, exponential, and 2-state Markov; the supported protocols are TCP, UDP, and ST-II.* The measurements, therefore, are from user process to user process. This tool is publicly available via anonymous ftp from sparkyfs.erg.sri.com.

2.3 MEASURED PARAMETERS

The experiments rely on measured parameters, i.e., metrics, to determine the effectiveness of the algorithm under study. These metrics, which are discussed in more detail below, currently include

- CPU idle time
- End-to-end per-packet delay
- End-to-end delay variance
- End-to-end packet loss
- End-to-end throughput
- Offer rate
- Offer load
- Fairness
- Path utilization
- Conversation-setup delay
- Blocking probability.

Each experiment described in this document is expected to use these definitions, and the suggested parameters below, in its execution.

*TCP: Transmission Control Protocol; UDP: User Datagram Protocol; ST-II: Stream Protocol, Version 2.

CPU idle time provides an indication of the complexity of the algorithm. If both the idle time and the link utilization are low, the router rather than the links is the bottleneck and thus the algorithm may be unsuitable. If both the idle time and the link utilization are high, the algorithm is extremely efficient and may be suitable for use in a high-speed network. The idle time may be measured via the UNIX command vmstat. For all experiments in this document, a measurement every 10 seconds is expected (vmstat 10) on each router involved in the experiment.

End-to-end per-packet delay, end-to-end delay variance, end-to-end packet loss, end-to-end throughput, offer rate, and **offer load** are computed from traffic traces. These traces may be from the traffic generator used in the experiment (see Subsection 2.2) or may be gathered using the program tcpdump, for example. **Offer rate** is the number of packets per second generated at the source; **offer load** is the bandwidth requested by the source.

Fairness is defined according to the max-min model developed by Bertsekas and Gallager.[*] This model tries to maximize the allocation of each user, subject to the constraint that an increase in one user's (i's) allocation does not cause a decrease in some other user's allocation that is already as small as i's or smaller. Figure 2 provides an example of a fair allocation according to this model. Fairness for a given conversation is computed by considering the throughput at the conversation's bottleneck link. The ratio of the largest throughput for any conversation to the throughput of that conversation at the bottleneck link is the measure of fairness for that conversation. Network-wide fairness is the average fairness for all conversations, weighted by each conversation's duration.

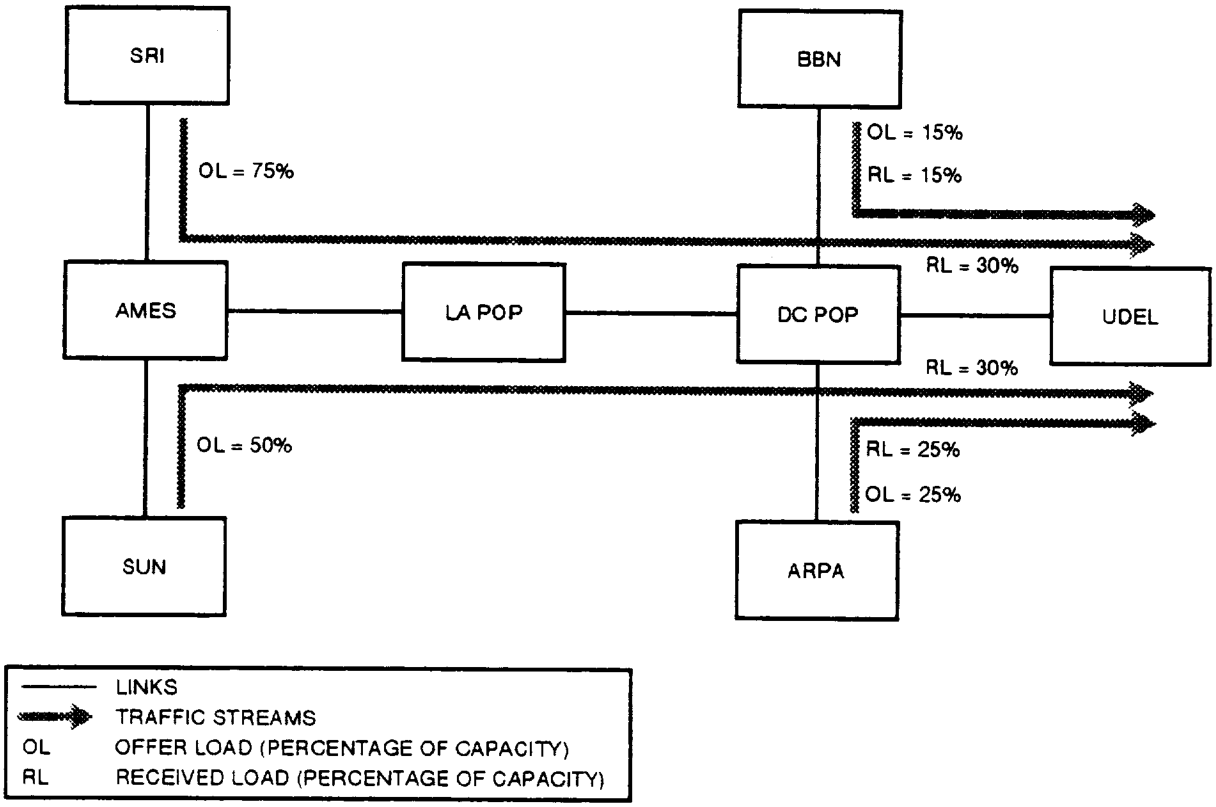


Figure 2. Max-Min Fair Solution

[*] Bertsekas, D., and Gallager, R. 1987. *Data Networks*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Path utilization for a given conversation is the overall utilization of its bottleneck link. As with fairness, network-wide path utilization is the average path utilization for all conversations, weighted by each conversation's duration.

Conversation-setup delay is the time starting when an application requests a reservation and ending when the request is either granted or denied. If the application chooses to block until the reservation can be honored, then there may be an additional conversation-blocking delay, which is the time starting when the request is initially denied and ending when the request is finally granted.

Blocking probability is the probability that a reservation request will not be immediately granted.

The following tables (Tables 1, 2, and 3) summarize the application of these metrics to the experiments found in Sections 3, 4, and 5.

Table 1. Best Effort Experiments

EXPERIMENT	DEFINITION	CPU IDLE TME	ETE DELAY	ETE DELAY VARIANCE	ETE PACKET LOSS	ETE THROUGHPUT	OFFER RATE	OFFER LOAD	FAIRNESS	PATH UTILIZATION	CONVERSATION-SETUP DELAY	BLOCKING PROBABILITY
Determining Overhead	Subsection 3.1	●	●		●		●	●				
Overload Behavior with Cooperative Sources	Subsection 3.2	●	●	●	●	●		●	●	●		
Overload Behavior with Uncooperative Sources	Subsection 3.3	●	●	●	●	●		●	●	●		
Delay for Low-Bandwidth User	Subsection 3.4	●	●	●	●	●		●	●	●		

Table 2. Type-of-Service Experiments

EXPERIMENT	DEFINITION	CPU IDLE TME	ETE DELAY	ETE DELAY VARIANCE	ETE PACKET LOSS	ETE THROUGHPUT	OFFER RATE	OFFER LOAD	FAIRNESS	PATH UTILIZATION	CONVERSATION-SETUP DELAY	BLOCKING PROBABILITY
Honoring Priorities	Subsection 4.1	●	●	●	●	●		●	●	●		
Honoring Delay Constraints	Subsection 4.2	●	●	●	●	●		●	●	●		
Honoring Throughput with Cooperative Sources	Subsection 4.3	●	●	●	●	●		●	●	●		
Honoring Throughput with Uncooperative Sources	Subsection 4.4	●	●	●	●	●		●	●	●		

Table 3. Resource Reservation Experiments

EXPERIMENT	DEFINITION	CPU IDLE TME	ETE DELAY	ETE DELAY VARIANCE	ETE PACKET LOSS	ETE THROUGHPUT	OFFER RATE	OFFER LOAD	FAIRNESS	PATH UTILIZATION	CONVERSATION-SETUP DELAY	BLOCKING PROBABILITY
Honoring Existing Reservations	Subsection 5.1	●	●	●	●	●		●		●	●	
Blocking Correlation	Subsection 5.2	●	●	●	●	●		●		●	●	●
Honoring Future Commitments	Subsection 5.3	●	●	●	●	●		●		●	●	

2.4 RESULTS

This section will contain a summary of the expected results and will note any conditions that may show a need to repeat the experiments.

3 EXPERIMENTS FOR BEST-EFFORT ALGORITHMS

The resource-allocation capabilities of best-effort algorithms are evaluated on the assumption that all users have equal rights to network resources. The experiments described in this section determine (1) the overhead, (2) the behavior of the algorithm in a saturated network with cooperative sources, (3) the behavior of the algorithm in a saturated network with some uncooperative sources, and (4) the qualitative evaluation of per-packet delay in a low-bandwidth conversation in a saturated network.

3.1 OVERHEAD DETERMINATION

The objective of this experiment is to evaluate the basic performance of the algorithm by finding the throughput of the network. The throughput is defined as the maximum amount of data per second that can be sent through a network while the delay remains approximately constant and no packet loss occurs.

3.1.1 Procedure

The experiment uses fixed-length packets, and increases the offer rate (at one packet per second) of a single UDP stream, until the network is overloaded. It is suggested that at each offer rate a minimum of 50,000 packets be sent. The network is considered to be overloaded when packets are consistently being dropped and the delay is greater than that achieved when there was no packet loss. Figure 3 shows the end-to-end delay for this experiment, to illustrate the states during the experiment and the point at which the throughput is established. Note that each interval marked on the x-axis indicates a new offer rate.

The packet sizes selected depend on the maximum transmission unit (MTU) of the network under test. It is suggested that the number of bytes in one packet not exceed that of the smallest MTU traversed by the stream's path. For example, in the case of an MTU of 1500 bytes, it is

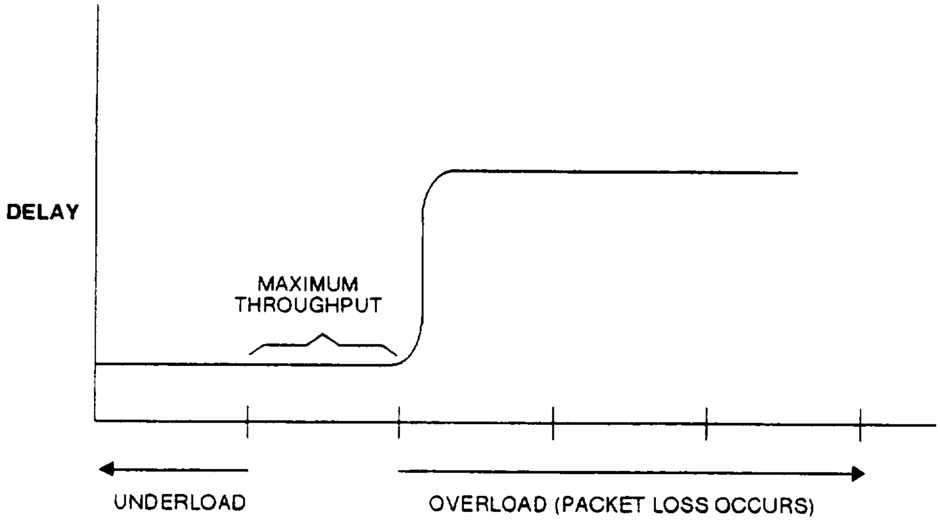


Figure 3. Determining Throughput

suggested that this experiment be performed with packet sizes of 250, 500, 750, 1250 and 1500 bytes. Note that if the TG is used, it is important to verify that the offer rate is correct and repeatable at small interarrival times, to ensure that the requested rate is provided and that rate is repeatable.

3.1.2 Measured Parameters

The parameters to be measured in the experiment are

- CPU idle time at gateways
- End-to-end per-packet delay
- End-to-end packet-loss rate
- Offer rate
- Offer load for each new offer rate.

3.1.3 Results

The results of this experiment are dependent on the speed of the networks and the implementation of the algorithm. However, some conditions indicate a need to rerun the experiment. If the packet-loss rate is significantly greater than expected, due to media error rates, then the point of underload has not been determined. The initial offer rate should be reduced until no anticipated loss is found, and the rate should then be incrementally increased from that point. It is anticipated that large packet sizes will obtain a higher throughput. The overhead of the algorithm can be determined by comparing the results against a widely used algorithm such as FIFO queueing.

3.2 OVERLOAD BEHAVIOR WITH COOPERATIVE SOURCES

The objective of this experiment is to evaluate the effect of a congestion-avoidance algorithm when the offered load exceeds capacity and the traffic sources are responding to evidence of congestion.

3.2.1 Procedure

The number of streams used in this experiment depends on the availability of hosts on the test network, and the ability of the test hosts to saturate the network bandwidth (a minimum of two streams is required). The experiment consists of all possible combinations of two different traffic distributions with two different start times (i.e., four separate tests). In one of the two traffic scenarios, all streams demand equal proportions; in the other scenario, all streams demand unequal proportions with at least one stream demanding a high percentage (95%, for example) of the link capacity. The aggregate bandwidth of all streams should exceed the bandwidth available on at least one of the paths. For each of these traffic scenarios, two different start times are used: one with all the streams beginning at the same time or one with the start times of the individual streams being staggered. When the start times are staggered, the times at which the individual streams start depend upon the length of the conversations and the number of streams involved. These times should also be such that the capacity of the network is exceeded during most of the experiment. The purpose of this kind of overloading is to test the ability of the algorithm to degrade gracefully.

To provide cooperative sources, a protocol that provides flow control should be used, e.g., TCP.

3.2.2 Measured Parameters

The parameters to be measured in this experiment are

- CPU idle time at gateways
- End-to-end per-packet delay per stream
- End-to-end delay variance per stream
- End-to-end packet loss per stream
- End-to-end throughput per stream
- Offer load per stream
- Fairness
- Path utilization.

3.2.3 Results

The results should indicate that the algorithm is fair: i.e., when all streams demand the same proportion but the sum of all streams is greater than the available capacity, each stream should receive an equal proportion. When one stream demands a high percentage of the bandwidth, the stream that is demanding less than its share should receive the entire amount, while the high-bandwidth stream receives whatever is left.* Staggering the start and end times should have no effect on the fairness of the algorithm.

3.3 OVERLOAD BEHAVIOR WITH UNCOOPERATIVE SOURCES

The objective of this experiment is to evaluate the effect of a congestion-avoidance algorithm when the offered load exceeds the network's capacity, and not all traffic sources are responding to evidence of congestion.

3.3.1 Procedure

The number of streams used in this procedure depends on the availability of hosts on the test network, and the ability of the test hosts to saturate the network bandwidth (a minimum of two streams is required). The experiment consists of all possible combinations of two different traffic distributions with two different start times (i.e., four separate tests). In one of the two traffic scenarios, all streams demand equal proportions; in the other scenario, all streams demand unequal proportions with at least one stream demanding a high percentage (95%, for example) of the link capacity. The aggregate bandwidth of all streams should exceed the bandwidth available on at least one of the paths. For each of these traffic scenarios, two different start times are used: one with all the streams beginning at the same time and one with the start times of the individual streams being staggered. When the start times are staggered, the times at which the individual streams start depend upon the length of the conversations and the number of streams involved. These times should also be such that the capacity of the network is exceeded during most of the experiment. The purpose of this kind of overloading is to test the ability of the algorithm to degrade gracefully.

To provide uncooperative sources, a protocol should be used that does not incorporate flow control, such as UDP.

*If more than 2 streams are used to lead the network, the bandwidth allocation should follow the max-min model described in Subsection 2.3.

3.3.2 Measured Parameters

The parameters to be measured in this experiment are:

- CPU idle time at gateways
- End-to-end per-packet delay per stream
- End-to-end delay variance per stream
- End-to-end packet loss per stream
- End-to-end throughput per stream
- Offer load per stream
- Fairness
- Path utilization.

3.3.3 Results

The results should be the same as those obtained with cooperative sources (see Subsection 3.2.3).

3.4 DELAY IN OVERLOADED NETWORK FOR LOW-BANDWIDTH CONVERSATION

The objective of this experiment is to evaluate the effect of a congestion-avoidance algorithm on the delay experienced by a single low-bandwidth conversation when the offered load exceeds the network's capacity and not all traffic sources are responding to evidence of congestion.

3.4.1 Procedure

The number of streams depends on the availability of hosts on the test network and the ability of the test hosts to saturate the network bandwidth. A minimum of two streams is required. The network is loaded with uncooperative sources, such as UDP streams, so that the offer rate of the competing streams exceeds their fair share, but the total number of streams is not so great as to cause the low-bandwidth conversation to use its fair share of the network resources. A qualitative evaluation of delay is provided by a human subject typing input to an echo server (e.g., a remotely executed UNIX cat command).

3.4.2 Measured Parameters

The parameters to be measured in this experiment are

- CPU idle time at gateways
- End-to-end per-packet delay per stream
- End-to-end delay variance per stream
- End-to-end packet loss per stream
- End-to-end throughput per stream
- Offer load per stream
- Fairness
- Path utilization.

Qualitative evaluation of delay for a single low-bandwidth conversation should be performed.

3.4.3 Results

The results of this experiment should show that the delay is constant: the flow of data should stay the same.

4 EXPERIMENTS FOR TYPE-OF-SERVICE ALGORITHMS

The resource-allocation capabilities of type-of-service algorithms are evaluated with the assumption that users have different rights to network resources, and that these differences are expressed in terms of priorities, delay constraints, and throughput limits. Algorithms that implement more complex types of service, such as combinations of throughput, delay, and priority, require more experiments to fully evaluate their capabilities and are therefore not included in CATE.

The experiments described in this section determine the ability of algorithms to honor (1) priorities, (2) delay constraints, (3) throughput limits with cooperative sources, and (4) throughput limits with uncooperative sources.

4.1 HONORING PRIORITIES

The objective of this experiment is to evaluate the ability of a congestion-avoidance algorithm to honor the priorities assigned to a subset of the offered load.

4.1.1 Procedure

The number of streams depends on the availability of hosts on the test network and on the ability of the test hosts to saturate the network bandwidth. A minimum of three streams is required: one high priority and two low priority. UDP streams should be used so that flow-control mechanisms within a transport protocol do not interfere with the outcome of the experiment. The aggregate throughput of all high-priority streams should not exceed the bandwidth of the paths; the low-priority streams are used to saturate the capacity of the network. For example, 60% of the available capacity should be assigned to high-priority streams; another 60% should be used by the low-priority streams. Since the throughput of the network is exceeded, this experiment also tests the fairness of the algorithm within the low-priority traffic class.

4.1.2 Measured Parameters

The parameters to be measured in this experiment are

- CPU idle time at gateways
- End-to-end per-packet delay per stream
- End-to-end delay variance per stream
- End-to-end packet loss per stream
- End-to-end throughput per stream
- Offer load per stream

- Fairness (within priority level)
- Path utilization, computed separately for each priority level; the computation of the value for a given priority level is based on all traffic that is at least as important as that priority level.

4.1.3 Results

The high-priority streams should receive all their traffic, with loss occurring only in the low-priority streams. The throughput of the low-priority streams should not be unfair: i.e., all low-priority streams should receive an equal proportion of the bandwidth remaining from the high-priority streams. The high-priority streams should also have an average delay lower than that of the low-priority streams for paths of equal length.

4.2 HONORING DELAY CONSTRAINTS

The objective of this experiment is to evaluate the ability of a congestion-avoidance algorithm to honor the delay constraints assigned to a subset of the offered load.

4.2.1 Procedure

The number of streams used depends on the availability of hosts on the test network and on the ability of the test hosts to saturate the network bandwidth. A protocol that does not provide flow control, such as UDP, should be used, so that there is no interaction with the behavior of the transport protocol. A subset of streams with a delay constraint is chosen to be serviced. If possible, streams with long and short paths should be included for traffic with delay constraints. The aggregate offered load of all streams should approach the capacity of the network. For each of the streams with a delay constraint, the minimum delay is determined by executing the stream in an unloaded network, i.e., by itself. The number of streams executing is incrementally increased until the point at which the increased delay-constrained load results in little or no increase in the amount of traffic reaching the destinations that meets the delay constraints. At this point, the traffic with no delay constraints is added. The experiment is repeated for each of these delay values per stream: (1) minimum delay, (2) twice the minimum delay, and (3) four times the minimum delay. If all the streams can be serviced with their delay constraints, the number of streams with delay constraints is increased until this is no longer true.

4.2.2 Measured Parameters

The parameters to be measured in this experiment are

- CPU idle time at gateways
- End-to-end per-packet delay per stream
- End-to-end delay variance per stream
- End-to-end packet loss per stream
- End-to-end throughput per stream
- Offer load per stream
- Fairness (within traffic class)
- Path utilization.

4.2.3 Results

The results of this experiment should verify that thc algorithm meets the delay constraints specified. The number of streams supported and the variance in delay indicate how well the algorithm provides this service.

4.3 HONORING THROUGHPUT LIMITS WITH COOPERATIVE SOURCES

The objective of this experiment is to evaluate the ability of a congestion-avoidance algorithm to honor the throughput limits assigned to a subset of the offered load, when the offered load complies with the reservation request.

4.3.1 Procedure

The number of streams used depends on the availability of hosts on the test network and the ability of the test hosts to saturate the network bandwidth. A protocol that does not provide flow control, such as UDP, should be used, so that there is no interaction with the behavior of the transport protocol. A subset of the offer load should have guaranteed throughput. The remaining streams should cause the aggregate throughput of all streams to exceed the capacity of the network. The guaranteed traffic should be cooperative: i.e., it should abide by its reservation request.

4.3.2 Measured Parameters

The parameters to be measured in this experiment are

- CPU idle time at gateways
- End-to-end per-packet delay per stream
- End-to-end delay variance per stream
- End-to-end packet loss per stream
- End-to-end throughput per stream
- Offer load per stream
- Fairness (within traffic class)
- Path utilization.

4.3.3 Results

The results of this experiment should verify that the algorithm meets the throughput constraints specified for the streams with guaranteed throughput limits. The remaining nonguaranteed traffic should be fair in its remaining allocation of the reserved capacity.

4.4 HONORING THROUGHPUT LIMITS WITH UNCOOPERATIVE SOURCES

The objective of this experiment is to evaluate the ability of a congestion-avoidance algorithm to honor the throughput limits assigned to a subset of the offered load, when the offered load exceeds its reservation request.

4.4.1 Procedure

The number of streams depends on the availability of hosts on the test network and the ability of the test hosts to saturate the network. A protocol that does not provide flow control, such as UDP, should be used, so that there is no interaction with the behavior of the transport protocol. A subset of the offer load should havc guaranteed throughput. The remaining streams should cause the aggregate throughput of all streams to exceed the capacity of the network. A subset of the guaranteed traffic should be uncooperative: i.e., it should exceed its reservation request, and the remaining guaranteed traffic should follow its request.

4.4.2 Measured Parameters

The parameters to be measured in this experiment are

- CPU idle time at gateways
- End-to-end per-packet delay per stream
- End-to-end delay variance per stream
- End-to-end packet loss per stream
- End-to-end throughput
- Offer load
- Fairness (within traffic class)
- Path utilization.

4.4.3 Results

The results of this experiment should verify that the algorithm meets the throughput constraints specified for the streams with guaranteed throughput limits that follow their reservation request. The guaranteed streams that exceed their reservation request should not prevent the nonguaranteed traffic from receiving its fair share, and should at least receive the throughput specified in their request. The remaining nonguaranteed traffic should be fair in its remaining allocation of the reserved capacity.

5 EXPERIMENTS FOR RESOURCE RESERVATION ALGORITHMS

The resource-allocation capabilities of resource reservation algorithms are evaluated through experiments that determine the ability of algorithms to (1) reject reservations that cannot be honored, (2) avoid blocking correlation, and (3) handle reservations made for a future time. The experiments in this section handle very simple reservations of throughput; algorithms that allow more complex reservations would require additional experiments to fully evaluate their additional capabilities. (The ability of an algorithm to enforce a reservation is evaluated by the experiments described in Section 4.)

5.1 OVERBOOKING

The objective of this experiment is to evaluate the ability of a reservation-based congestion-avoidance algorithm to reject reservations that cannot be honored because of a lack of network resources. These reservation requests are to be immediately serviced.

5.1.1 Procedure

The number of streams used depends on the availability of hosts on the test network and on the ability of the test hosts to saturate the network bandwidth. Reservations are made between selected pairs of hosts with random bandwidth requirements and a random duration between 3 and 10 minutes. The traffic streams should be steady in their offered load. The aggregate offered load should approach the maximum throughput determined by the experiments described in Section 3. Additional conversations that oversubscribe the network resources should then be added at random intervals. As conversations end, new conversations should then be accepted.

5.1.2 Measured Parameters

The parameters to be measured in this experiment are

- CPU idle time at gateways
- Throughput per conversation
- Path utilization
- Offer load per conversation
- Conversation-setup delay
- End-to-end per-packet delay
- End-to-end delay variance
- End-to-end packet loss.

5.1.3 Results

Attempted overbooking should result in the rejection of one or more streams. The throughput of streams currently being serviced should not be affected by new requests. As time progresses, new reservation requests should be granted in a timely manner as original conversations end. It is expected that a reservation-based algorithm will be able to make efficient use of network resources (evidenced by high path utilizations), and also will be able to avoid overcommitting resources (evidenced by throughputs close to the reserved amounts, and low loss).

5.2 BLOCKING CORRELATION

The objective of this experiment is to determine how conversation-setup delay and conversation-blocking probability correlate with conversation attributes in a saturated network.

5.2.1 Procedure

The number of streams used depends on the availability of hosts on the test network and on the ability of the test hosts to saturate the network bandwidth. Reservations are made with random bandwidth requirements between selected pairs of hosts, with a random duration between 3 and 10 minutes selected uniformly. These reservations are made between hosts that contain long and short paths to the destinations. New reservation requests are also made randomly during the 3- to 10-minute intervals. The aggregate throughput of all the requests should exceed the network capacity. The traffic streams should be steady in their offered load.

5.2.2 Measured Parameters

The parameters to be measured in this experiment are

- CPU idle time at gateways
- Throughput per conversation
- Path utilization
- Offer load per conversation
- Conversation blocking probability
- Conversation-setup delay
- End-to-end per-packet delay
- End-to-end delay variance
- End-to-end packet loss.

5.2.3 Results

It is expected that a reservation-based algorithm will not exhibit extreme correlations; for example, long-path reservations should not be completely starved by short-path reservations. It is also expected that continuing to send large numbers of reservation requests will not significantly affect existing connections.

5.3 HONORING FUTURE COMMITMENTS

The objective is to determine how well a reservation-based algorithm takes future commitments (such as scheduled teleconferences) into account.

5.3.1 Procedure

The number of streams used depends on the availability of hosts on the test network and on the ability of the test hosts to saturate the network bandwidth. Reservations are made with random bandwidth requirements between selected pairs of hosts, with a random duration between 3 and 10 minutes selected uniformly. A subset of traffic forms a future traffic schedule. The conversations are due to start within the 3- to 10-minute interval and persist for a random duration of 3 to 10 minutes. These future commitments start with a random delay from the start time of the reservation; this delay of 10 to 60 seconds is selected uniformly. The traffic should be sufficiently high that the aggregate throughput of all streams exceeds the capacity of the network. All traffic streams should be steady in their offered load.

5.3.2 Measured Parameters

The parameters to be measured in this experiment are

- CPU idle time at gateways
- Throughput per conversation
- Path utilization

- Offer load per conversation
- Conversation-setup delay
- End-to-end per-packet delay
- End-to-end delay variance
- End-to-end packet loss.

5.3.3 Results

It is expected that a reservation-based algorithm that does allow for future commitments will deny reservation requests that conflict with a future commitment. No accepted reservation should be terminated prematurely or receive less than its reservation because of the activation of a future commitment.

6 SUMMARY

This manual has outlined a series of experiments that will aid researchers in evaluating the resource allocation and reservation capabilities of three different kinds of algorithms: best effort, type of service, and resource reservation. The best-effort experiments determine (1) the overhead, (2) the behavior of the algorithm in a saturated network with cooperative sources, (3) the behavior of the algorithm in a saturated network with some uncooperative sources, and (4) the qualitative evaluation of per-packet delay in a low-bandwidth conversation in a saturated network. The type-of-service experiments determine the ability of algorithms to honor (1) priorities, (2) delay constraints, (3) throughput limits with cooperative sources, and (4) throughput limits with uncooperative sources. The resource-reservation experiments determine the ability of algorithms to (1) reject reservations that cannot be honored, (2) avoid blocking correlation, and (3) handle reservations made for a future time.

These tests can be used to make meaningful comparisons between algorithms of the types described above. Furthermore, no assumptions are made regarding the underlying network, so that these tests can be performed in almost any environment; and the suggested tools for creating the traffic load make the experiments repeatable. In the past, it was very difficult to obtain reliable results, because duplicating the scenario was difficult. These experiments, therefore, form a set of benchmarks for evaluating the behavior of various networking algorithms.

Appendix C

BASELINE MEASUREMENTS

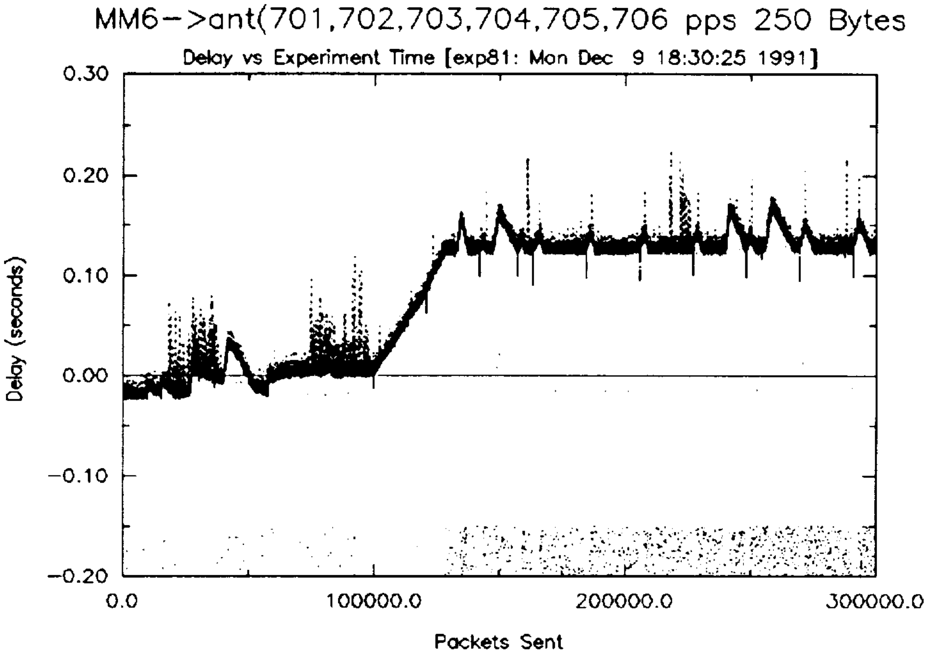
BASELINE MEASUREMENTS

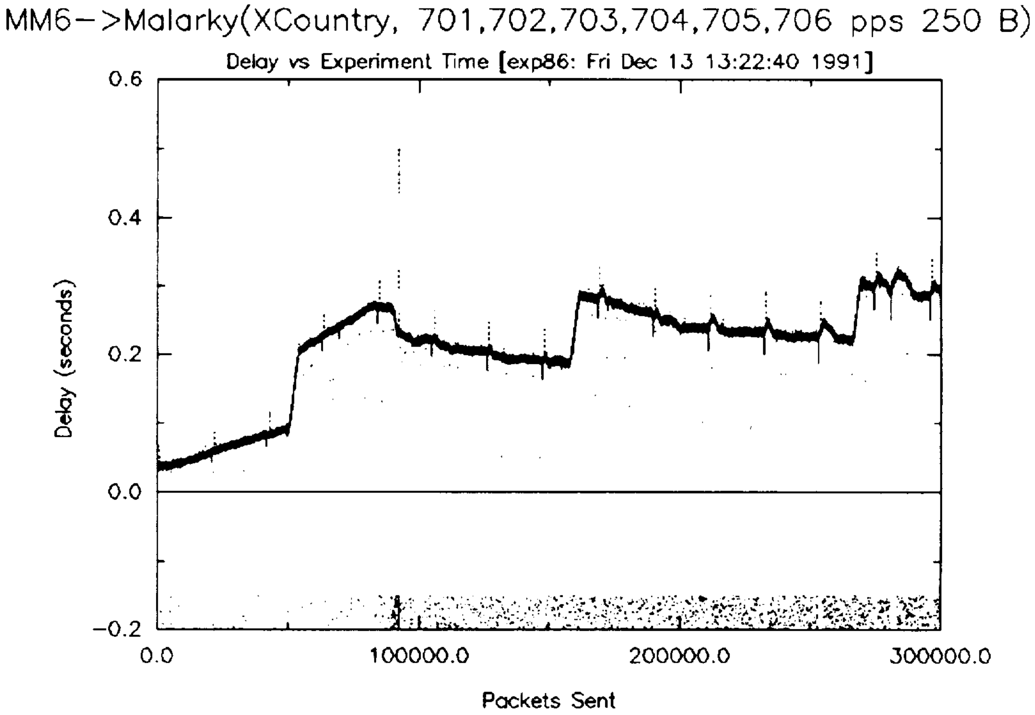
The graphs in Subsections C.1, C.2, and C.3 show the delay and loss for different offer rates. These graphs were used to determine the maximum throughput of DARTnet. The loss is indicated as a scatter diagram below the X axis. The source, destination, offer rate and packet byte count are indicated in the subtitles for each graph. Subsection C.4 contains similar graphs, but these graphs are examples of blackouts experienced during the execution of various experiments.
Subsection C.4 also contains an example of an experiment in which out-of-order delivery of packets occurred. The textual output for this example is from the log file produced by the Traffic Generator.

C.1 DARTnet THROUGHPUT

The figures in Subsection C.1 show throughput on the DARTnet at a line speed of 1.536 Mb/s. Service was provided by PSI.

THROUGHPUT FOR 250-BYTE PACKETS (1.404 Mb/s)

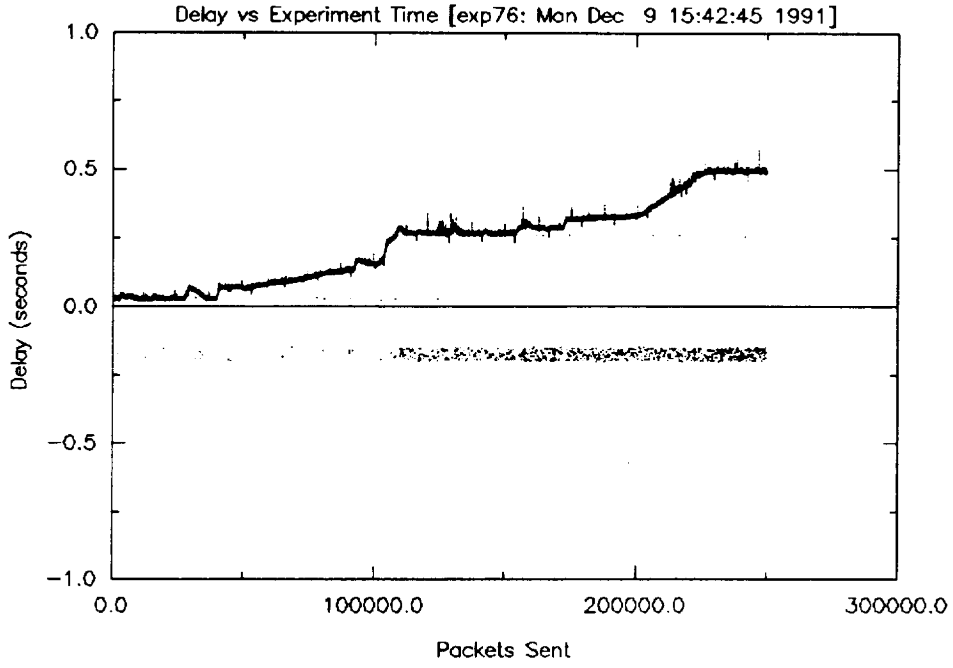




THROUGHPUT FOR 435-BYTE PACKETS (1.454 Mb/s)

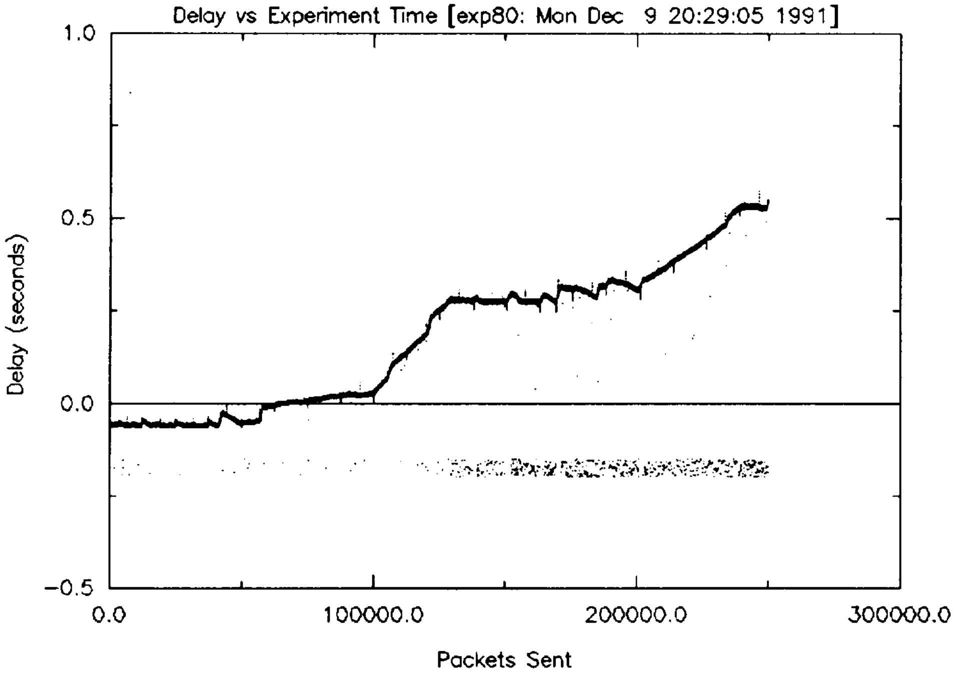
MM6->ant (418,419,420,421,422 pps, 435 Bytes)

Delay vs Experiment Time [exp76: Mon Dec 9 15:42:45 1991]

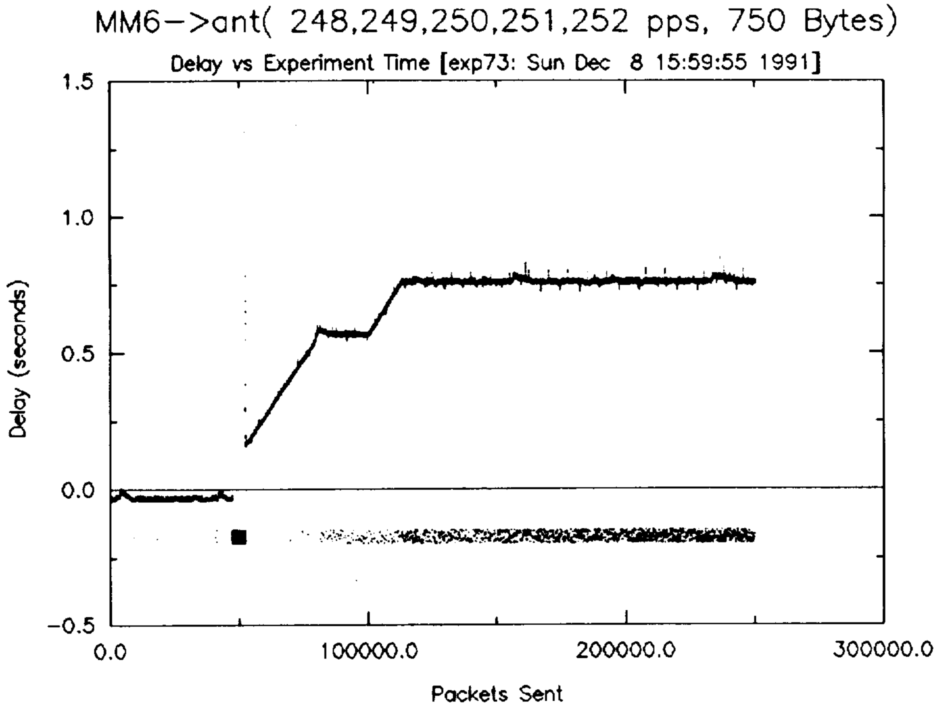


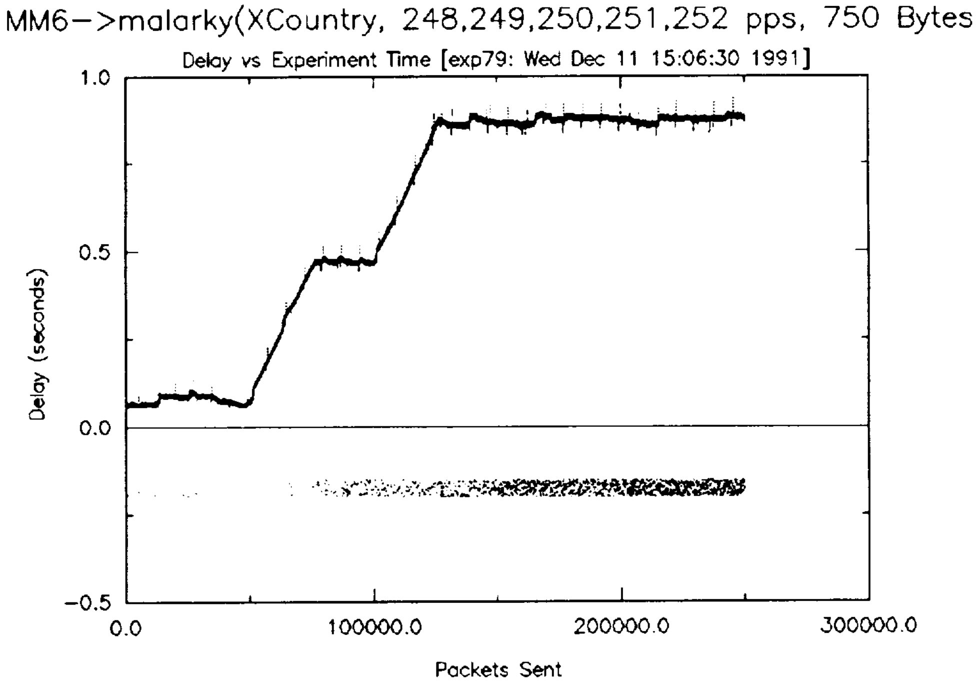
MM6->Malarky(XCountry, 418,419,420,421,422 pps 435 Bytes

Delay vs Experiment Time [exp80: Mon Dec 9 20:29:05 1991]

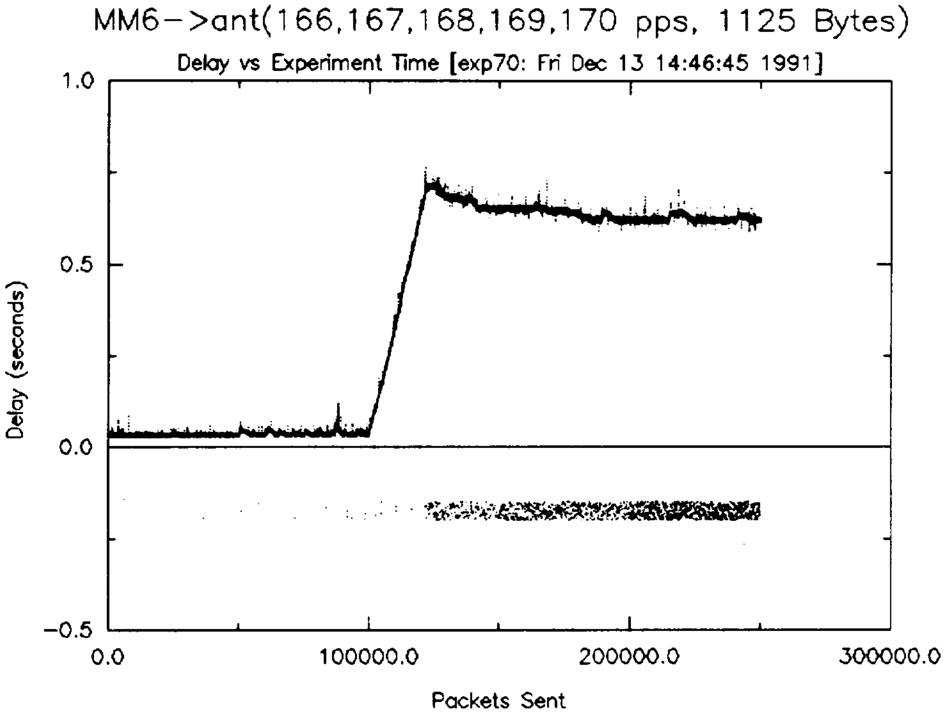


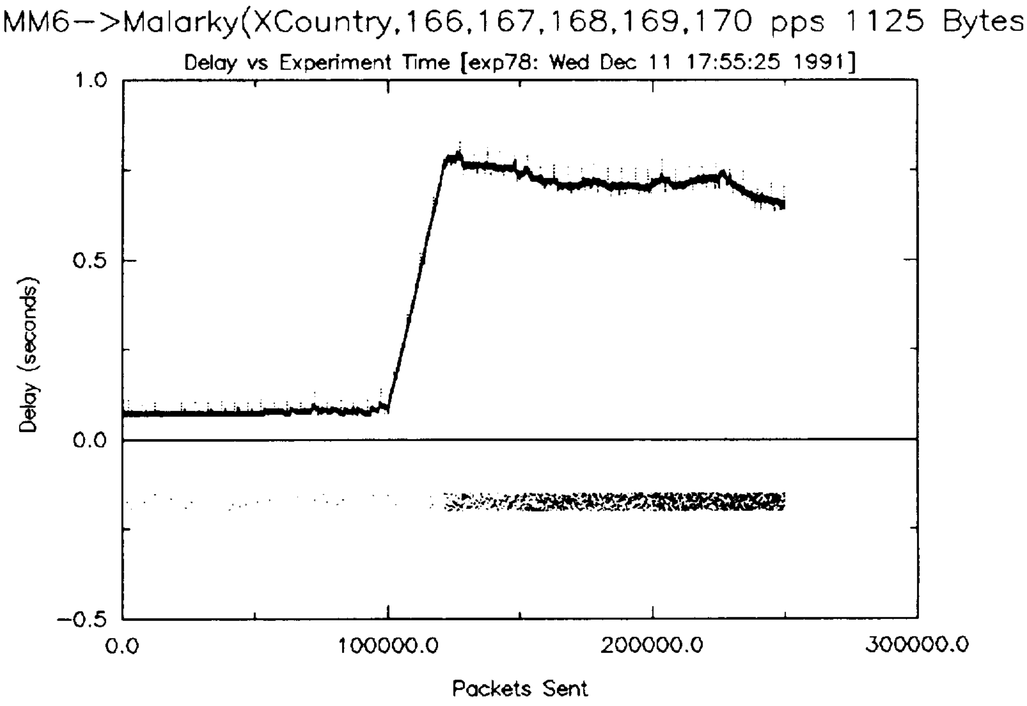
THROUGHPUT FOR 750-BYTE PACKETS (1.488 Mb/s)



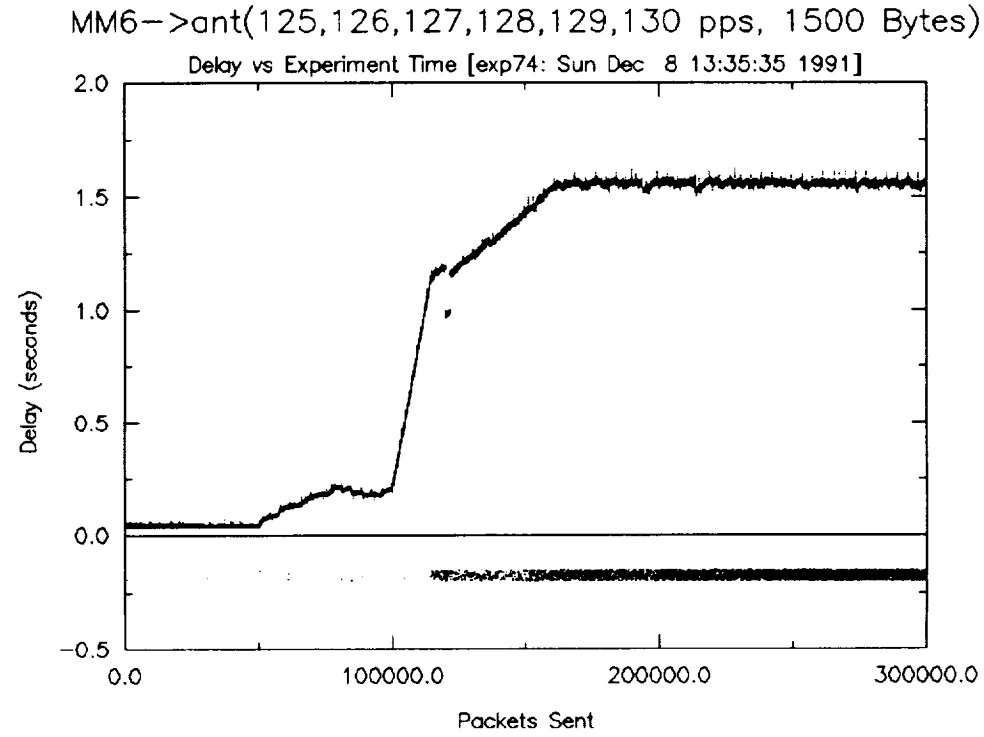


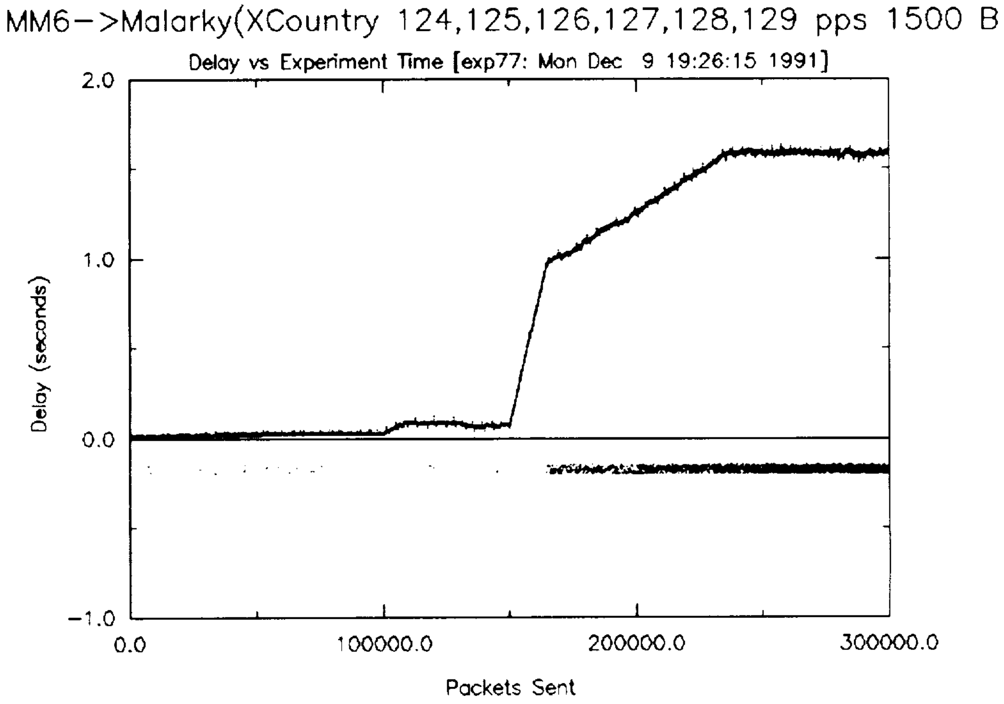
THROUGHPUT FOR 1125-BYTE PACKETS (1.503 Mb/s)





THROUGHPUT FOR 1500-BYTE PACKETS (1.500 Mb/s)

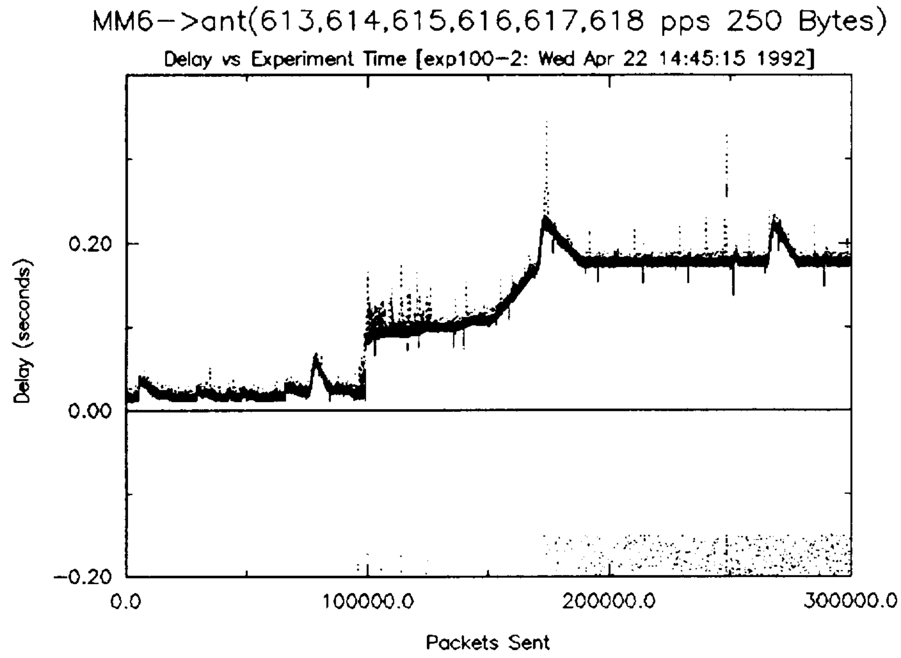


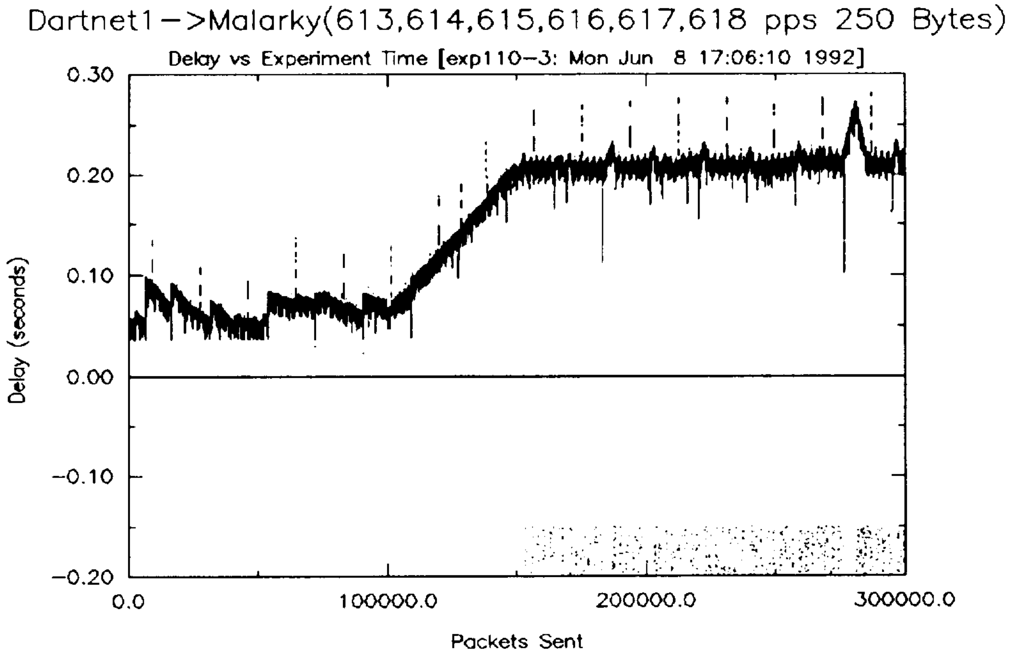


C.2 DARTnet THROUGHPUT

The figures in Subsection C.2 show DARTnet throughput at a line speed of 1.344 Mb/s. Service was provided by PSI.

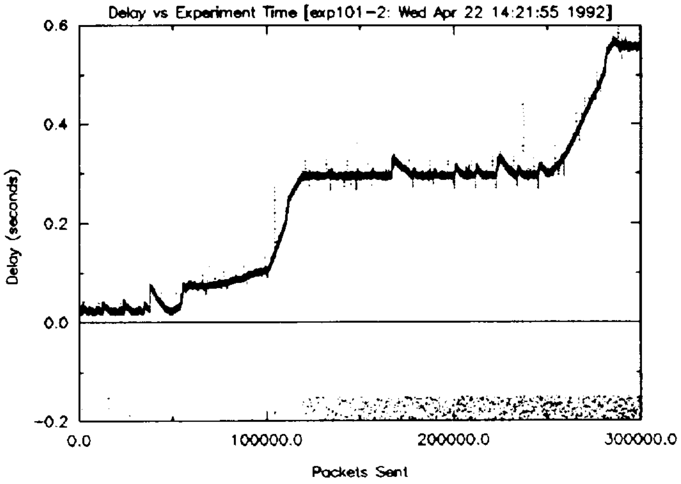
THROUGHPUT FOR 250-BYTE PACKETS (1.230 Mb/s)



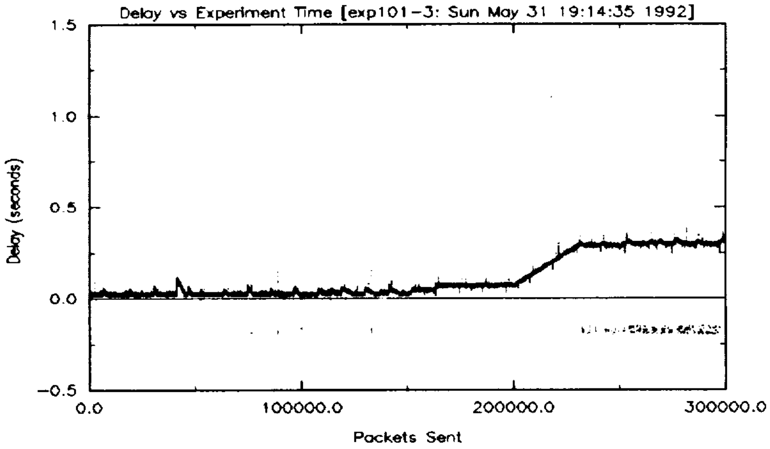


THROUGHPUT FOR 435-BYTE PACKETS (1.274 Mb/s)

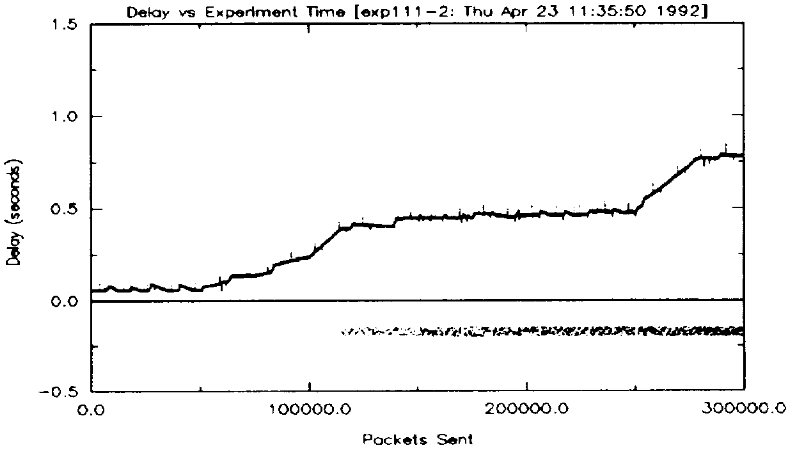
MM6->ant(366,367,368,369,370,371 pps 435 Bytes)



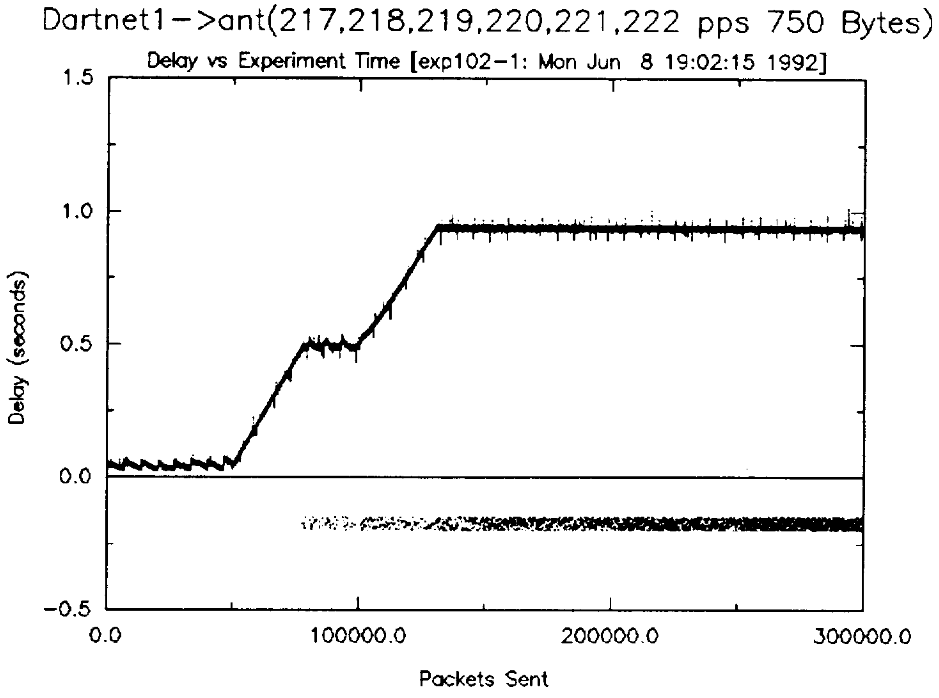
Dartnet1->ant(366,367,368,369,370,371 pps 435 Bytes)



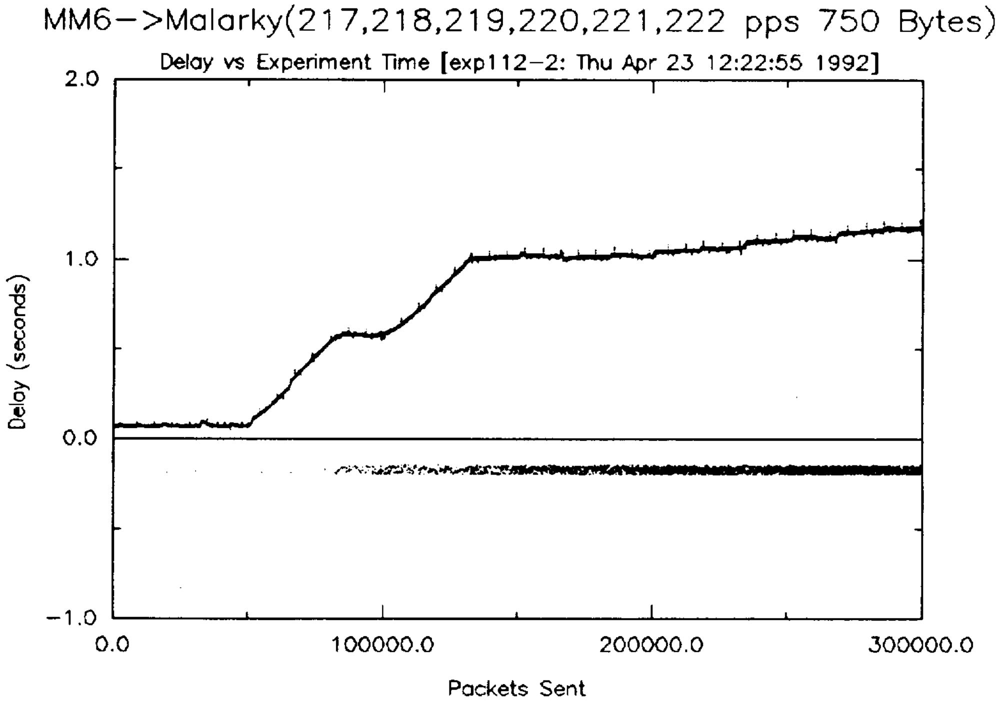
MM6->Malarky(366,367,368,369,370,371 pps 435 Bytes)



THROUGHPUT FOR 750-BYTE PACKETS (1.302 Mb/s)

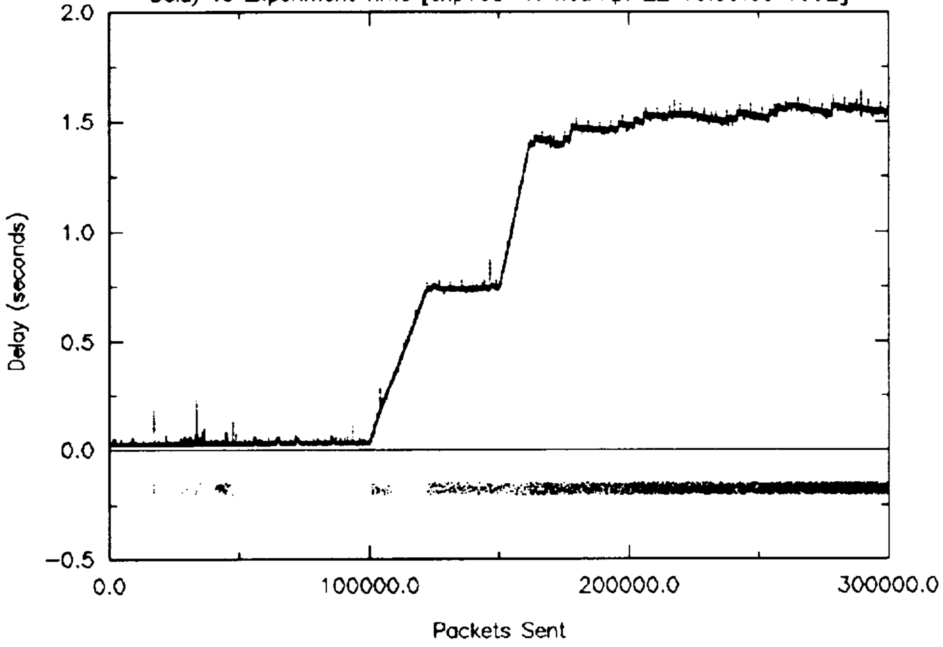

Dartnet1−>ant(217,218,219,220,221,222 pps 750 Bytes)

Delay vs Experiment Time [exp102−1: Mon Jun 8 19:02:15 1992]


MM6−>Malarky(217,218,219,220,221,222 pps 750 Bytes)

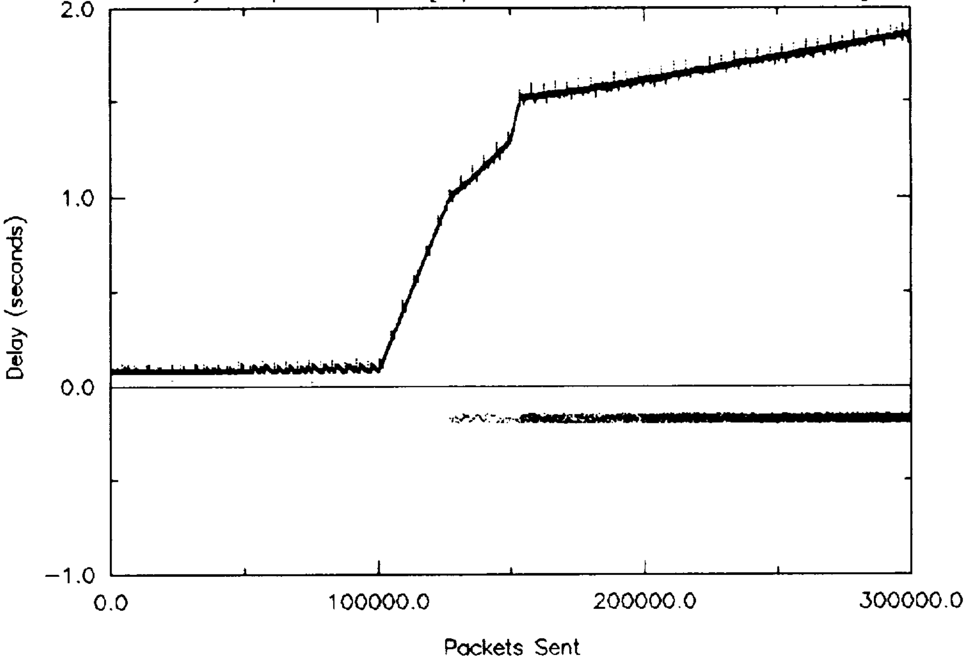
Delay vs Experiment Time [exp112−2: Thu Apr 23 12:22:55 1992]

THROUGHPUT FOR 1125-BYTE PACKETS (1.314 Mb/s)



MM6->ant(145,146,147,148,149,150 pps 1125 Bytes)

Delay vs Experiment Time [exp103-1: Wed Apr 22 10:39:05 1992]



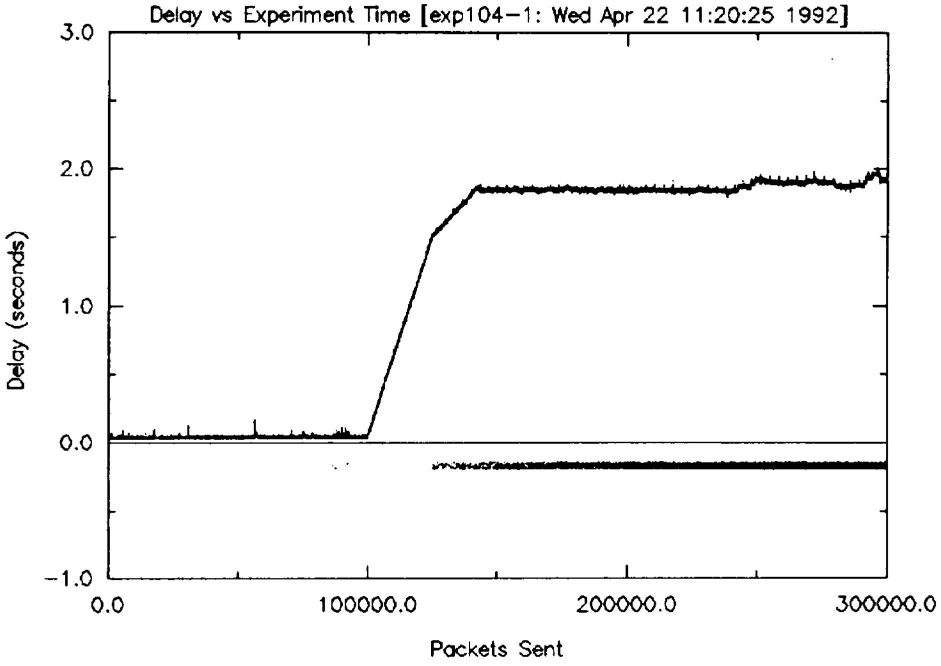
Dartnet1->Malarky(145,146,147,148,149,150 pps 1125 Bytes)

Delay vs Experiment Time [exp113-3: Mon Jun 8 16:28:20 1992]

THROUGHPUT FOR 1500-BYTE PACKETS (1.320 Mb/s)

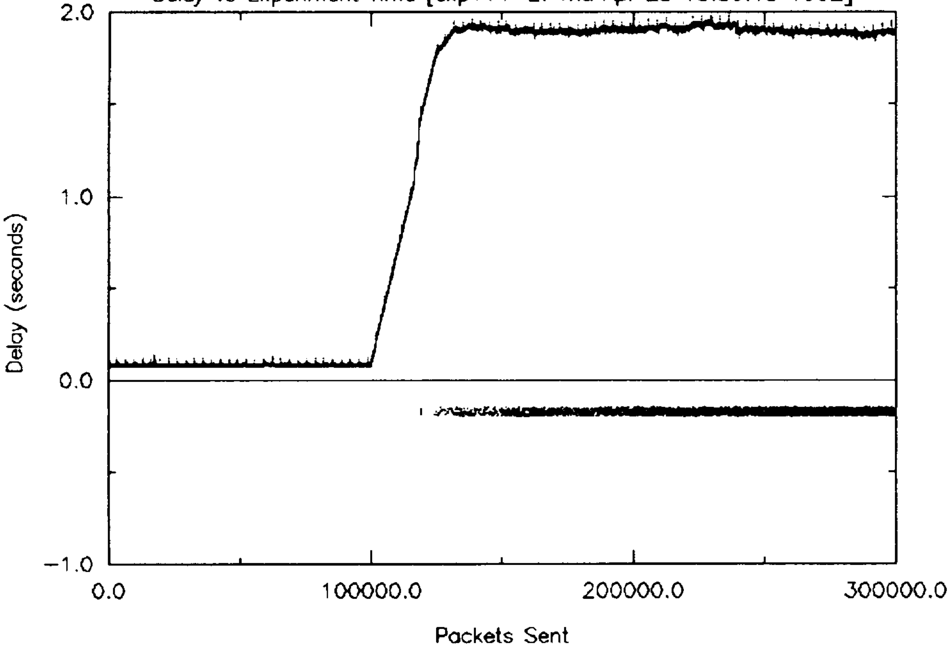
MM6−>ant(109,110,111,112,113,114 pps 1500 Bytes)

Delay vs Experiment Time [exp104−1: Wed Apr 22 11:20:25 1992]



MM6−>Malarky(109,110,111,112,113,114 pps 1500 Bytes)

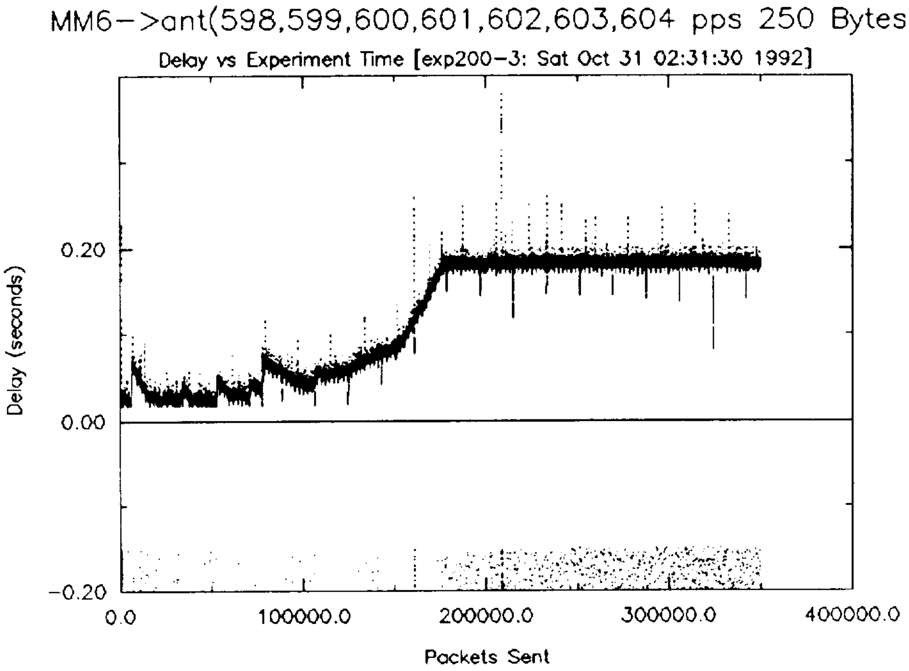
Delay vs Experiment Time [exp114−2: Thu Apr 23 13:59:15 1992]

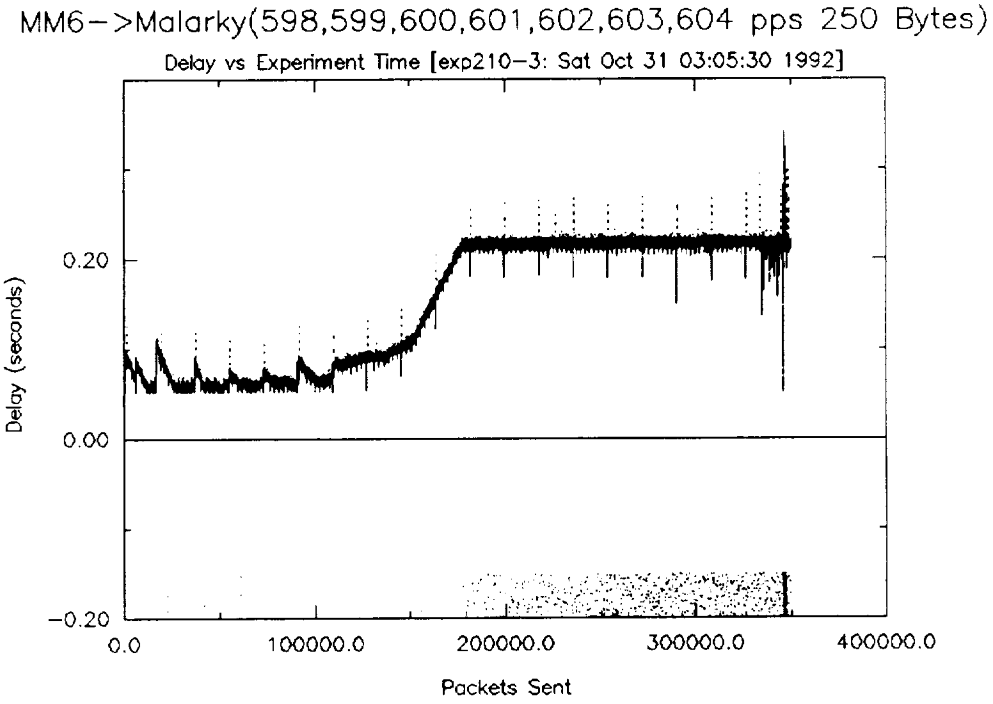


C.3 DARTnet THROUGHPUT

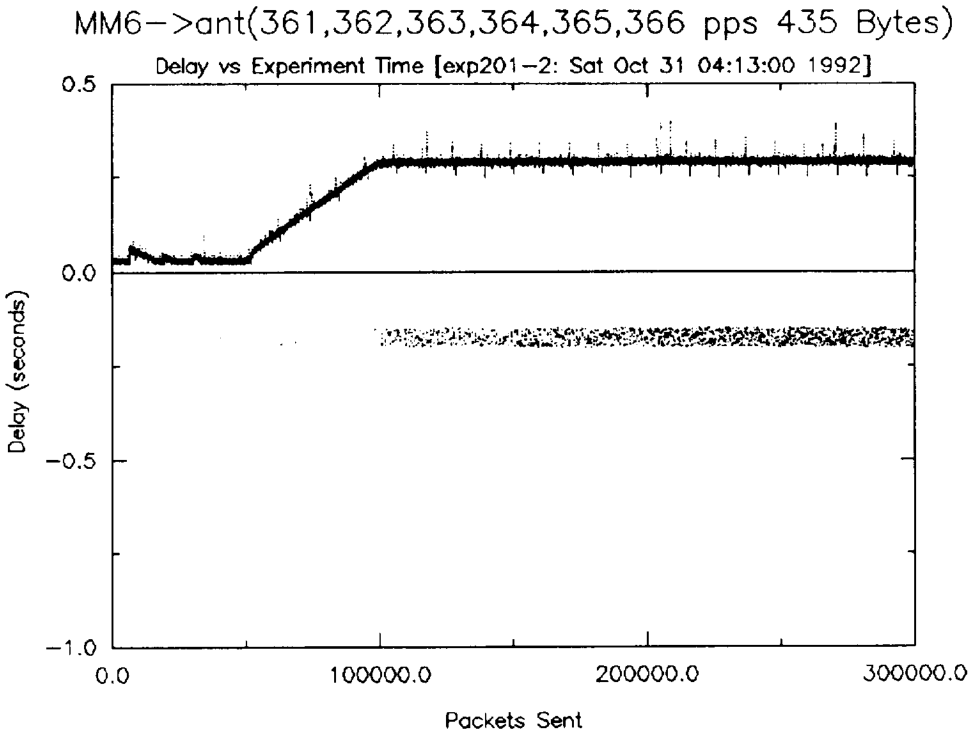
The figures in Subsection C.3 show DARTnet throughput at a line speed of 1.344 Mb/s. Service was provided by Sprint.

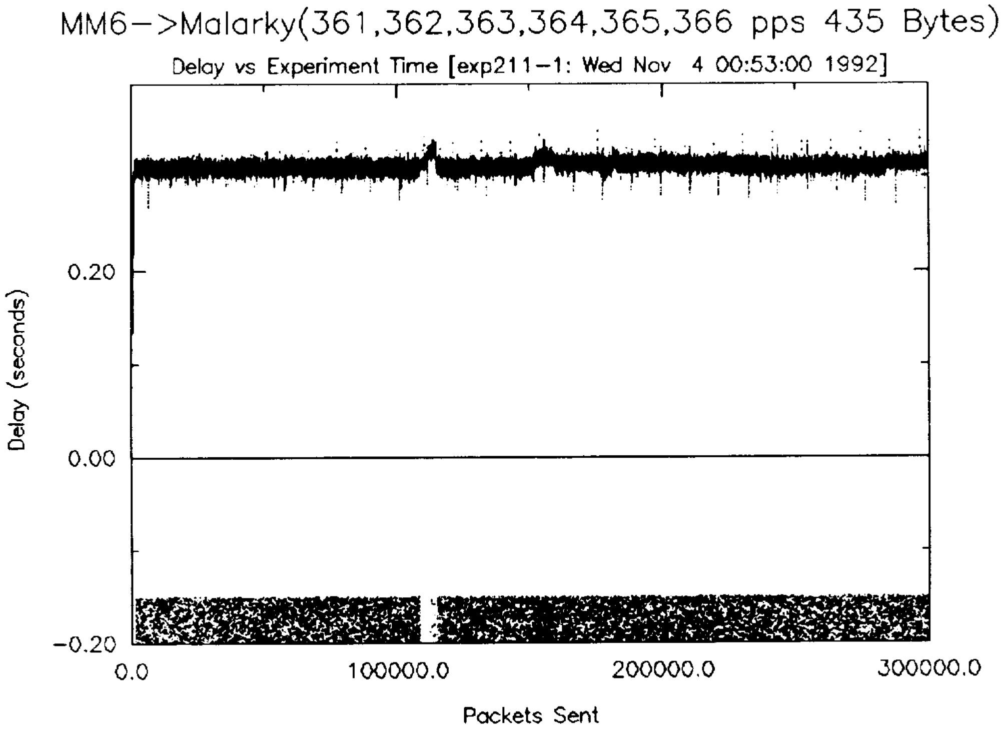
THROUGHPUT 250-BYTE PACKETS (1.198 Mb/s)



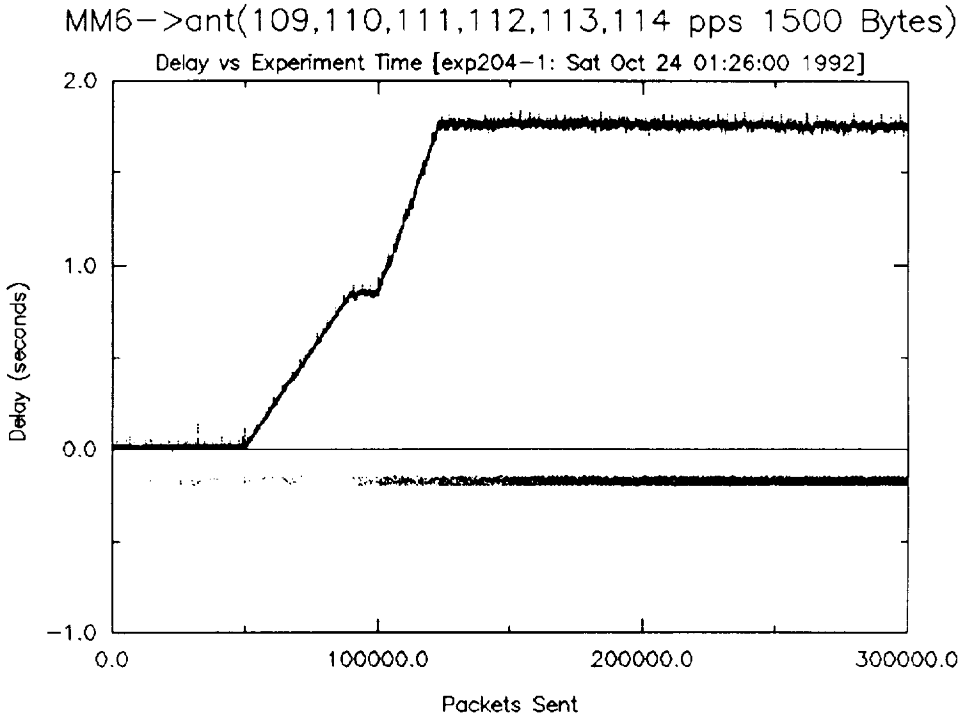


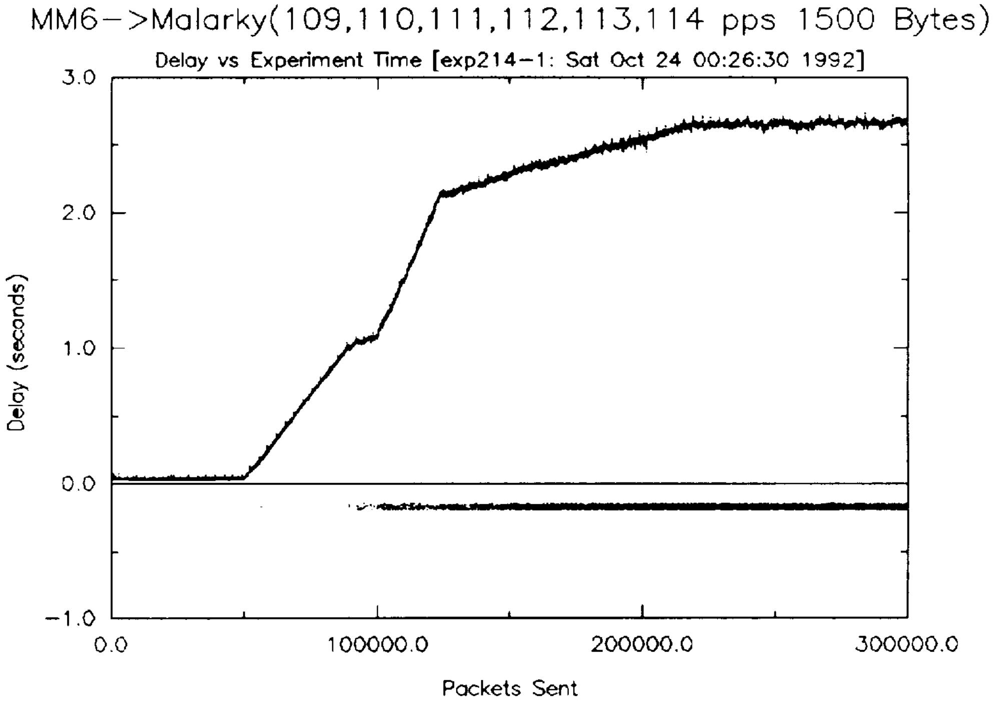
THROUGHPUT FOR 435-BYTE PACKETS (1.256 Mb/s)





THROUGHPUT FOR 1500-BYTE PACKETS (1.308 Mb/s)

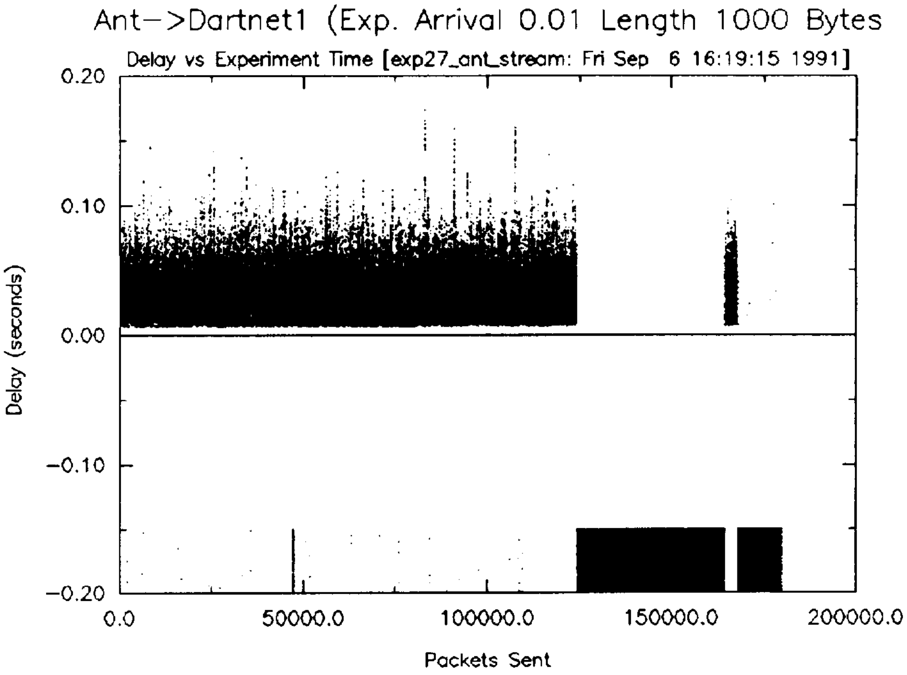


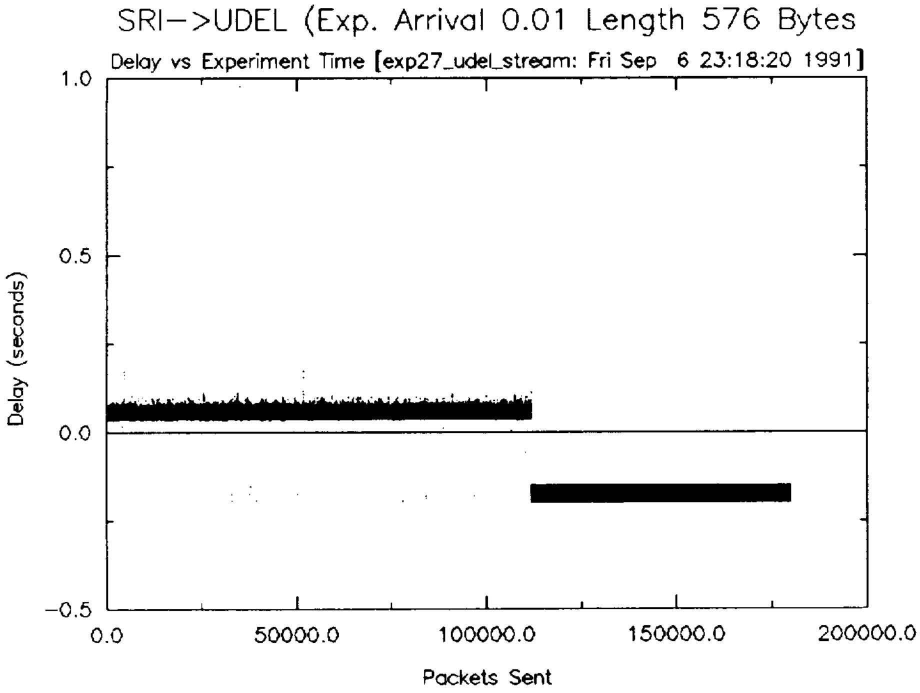


C.4 BAD NETWORK BEHAVIOR

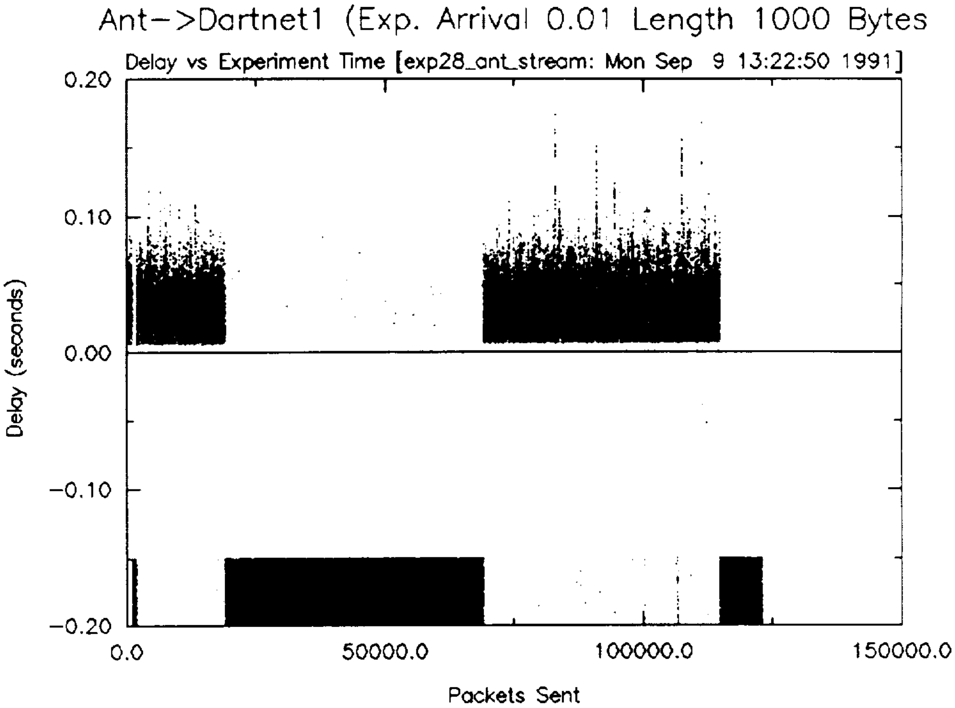
The figures in Subsection C.4 show examples of blackouts and out-of-order delivery during various tests.

BLACKOUTS DURING TWO-STREAM TESTS

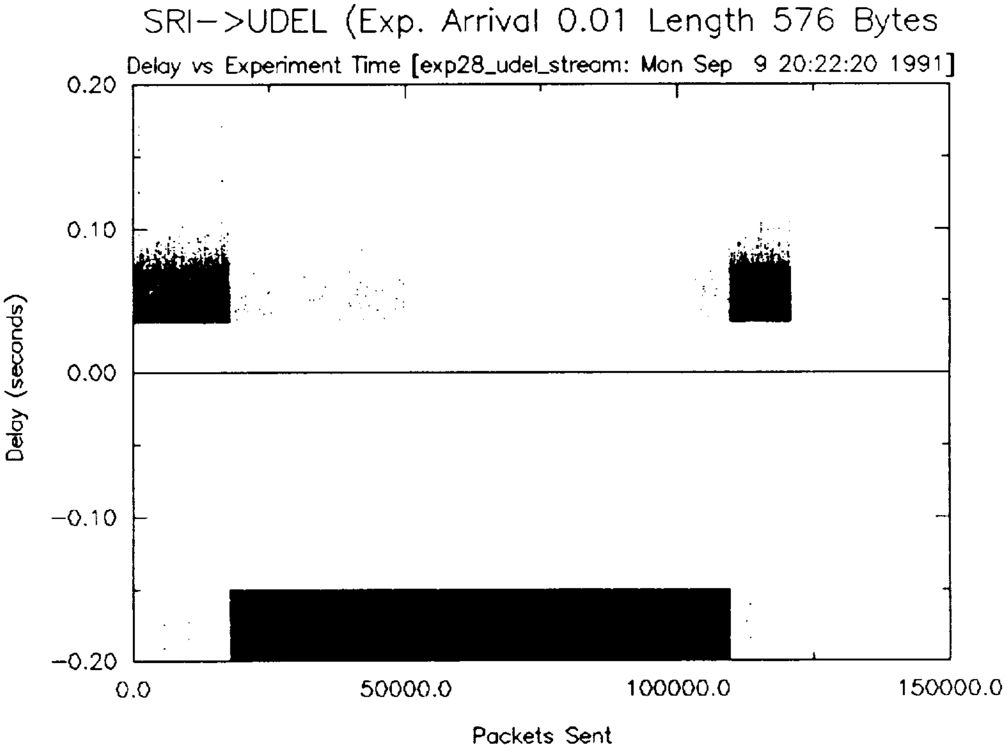




BLACKOUTS DURING TWO-STREAM TESTS

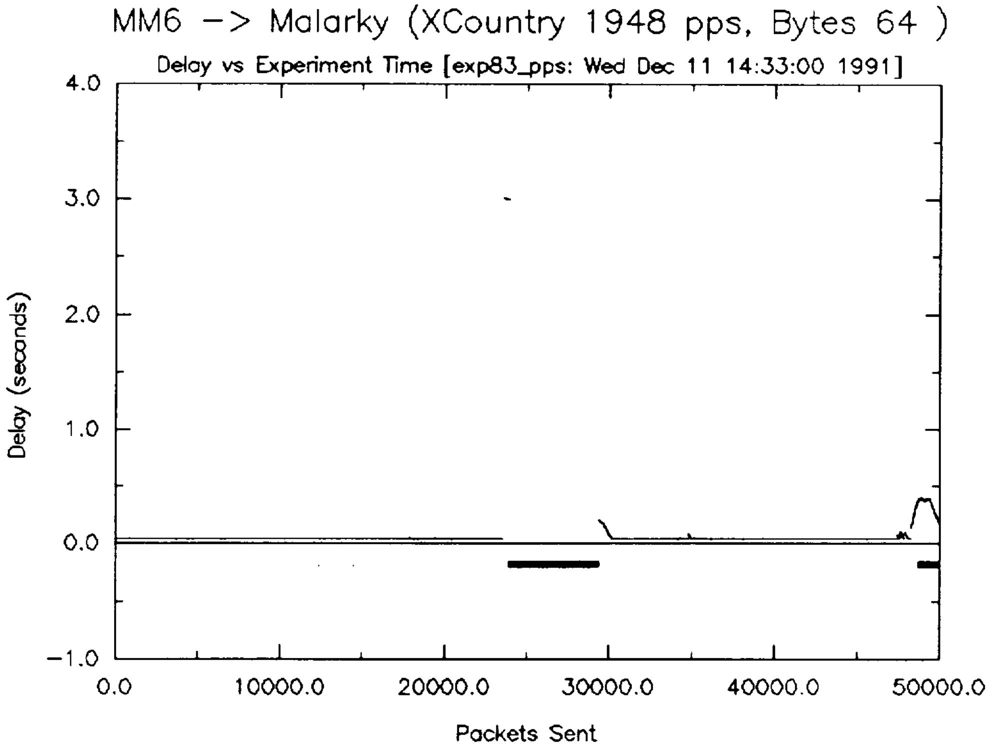

Ant->Dartnet1 (Exp. Arrival 0.01 Length 1000 Bytes

Delay vs Experiment Time [exp28_ant_stream: Mon Sep 9 13:22:50 1991]


SRI->UDEL (Exp. Arrival 0.01 Length 576 Bytes

Delay vs Experiment Time [exp28_udel_stream: Mon Sep 9 20:22:20 1991]

BLACKOUT WHILE NETWORK WAS UNDERLOADED



OUT-OF-ORDER DELIVERY EXAMPLE (TG Log File)

```
EXP80
CLIENT
141.374476 Transmit 128.4.0.22.2345 52828 403
141.375096 Transmit 128.4.0.22.2345 52829 403
141.375720 Transmit 128.4.0.22.2345 52830 403
141.376340 Transmit 128.4.0.22.2345 52831 403
141.383860 Transmit 128.4.0.22.2345 52832 403
141.384472 Transmit 128.4.0.22.2345 52833 403
141.385091 Transmit 128.4.0.22.2345 52834 403
141.393953 Transmit 128.4.0.22.2345 52835 403
141.394571 Transmit 128.4.0.22.2345 52836 403
141.395198 Transmit 128.4.0.22.2345 52837 403
141.395816 Transmit 128.4.0.22.2345 52838 403
141.396436 Transmit 128.4.0.22.2345 52839 403
141.403859 Transmit 128.4.0.22.2345 52840 403
141.404472 Transmit 128.4.0.22.2345 52841 403
141.405097 Transmit 128.4.0.22.2345 52842 403
141.405723 Transmit 128.4.0.22.2345 52843 403
141.413861 Transmit 128.4.0.22.2345 52844 403
141.414478 Transmit 128.4.0.22.2345 52845 403

SERVER
141.324255 Receive 140.173.160.3.2848 52828 403
141.326653 Receive 140.173.160.3.2848 52829 403
141.329012 Receive 140.173.160.3.2848 52830 403
141.331411 Receive 140.173.160.3.2848 52831 403
141.333793 Receive 140.173.160.3.2848 52832 403
141.336200 Receive 140.173.160.3.2848 52833 403
141.338557 Receive 140.173.160.3.2848 52838 403
141.340941 Receive 140.173.160.3.2848 52835 403
141.343566 Receive 140.173.160.3.2848 52836 403
141.345733 Receive 140.173.160.3.2848 52837 403
141.348113 Receive 140.173.160.3.2848 52839 403
141.350494 Receive 140.173.160.3.2848 52840 403
141.352888 Receive 140.173.160.3.2848 52841 403
141.355310 Receive 140.173.160.3.2848 52842 403
141.357680 Receive 140.173.160.3.2848 52843 403
141.360127 Receive 140.173.160.3.2848 52844 403
141.362512 Receive 140.173.160.3.2848 52845 403
```

Appendix D

REPORT ON STOCHASTIC FAIRNESS QUEUEING EXPERIMENTS

Technical Report • March 1993

A REPORT ON STOCHASTIC FAIRNESS QUEUEING (SFQ) EXPERIMENTS

Barbara A. Denny, Computer Scientist
Information and Telecommunications Sciences Center

Project 8600
ITAD-8600-TR-93-62

Prepared for:

NASA Ames Research Center
Moffett Field, California 94035

Attn: Dr. Henry Lum, Code RI, M/S: 244-7

and

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, Virginia 22203-1714

Attn: Dr. Paul Mockapetris

Approved by:

Boyd C. Fair, Director
Information and Telecommunications Sciences Center

Michael S. Frankel, Vice President and Director
Information, Telecommunications, and Automation Division

CONTENTS

FIGURES

TABLES

1 EXECUTIVE SUMMARY

SRI International (SRI) has developed an improved queueing algorithm, known as Stochastic Fairness Queueing (SFQ), for best-effort traffic (i.e., traffic that does not require any guaranteed service). SFQ is a probablistic variant of strict fair queueing where instead of a single queue being allocated per flow, a fixed number of queues are used and a hash function maps the IP source and destination to a particular queue. A seed to the hash function is also perturbed occasionally to help distribute the flows amongst different queues when more than one flow maps to the same queue during the lifetime of the flow. SFQ provides "fair" access by trying to ensure that each flow from source to destination host obtains equal access to the available bandwidth.

This report covers a series of experiments performed on DARTnet* evaluating the behavior and performance of SFQ against a FIFO queueing discipline. These experiments were designed to show SFQ's advantages and performance, and include tests demonstrating

- Fair utilization of available resources
- Starvation prevention
- Graceful degradation under overload conditions
- Resource usage.

The details of each experiment, including objective, procedures, data, and results, are presented in Section 4.

In general, the experiments do show that SFQ is better than FIFO queueing at allocating bandwidth equally among a set of flows. SFQ also prevents a stream from dominating the available bandwidth, which seems to be a tendency with FIFO queueing (i.e., if a flow demands more than its share of the available bandwidth, with FIFO queueing that stream receives a disproportionate amount when compared to flows demanding less than their share). Furthermore, SFQ seems to reward "nice" users of the network by providing a lower variance in delay and more throughput when their resource demand is less than their available share. Both SFQ and FIFO queueing seem to degrade fairly well as the network becomes saturated and to recover well as the network becomes less congested. Not unexpectedly, FIFO queueing is a little more efficient than SFQ—the delays are less and the throughput slightly higher because SFQ requires more processing. However, the performance difference between the two queueing disciplines is relatively small.

However, the experiments do point out some interesting behavior. FIFO queueing can behave better than SFQ with seed perturbation. We recommend further evaluation of the hash function and the seed perturbation technique. There are probably weaknesses in their current selection that cause this unexpected behavior. SFQ also seems to possess good scaling properties. To verify this, more experiments with a larger number of streams from more hosts need to be executed and examined, including the staggered introduction of streams. Staggering the streams may prove important, because graphs in the degradation experiment revealed some unexpected increases and decreases in throughput, which should be examined. This may again be due to the interaction of the hash function with the seed perturbation but it may also be related to some other unknown problem.

*DARTnet is a T1 testbed network sponsored by DARPA.

2 INTRODUCTION

This report summarizes a set of experiments comparing first-in, first-out (FIFO) queueing and Stochastic Fairness Queueing (SFQ). Historically, FIFO queueing has been the discipline in general use in routers. A single queue is used for all packets, which are serviced in a first-come, first-served manner. However, FIFO queueing often exhibits unfair behavior in the presence of multiple streams, especially during times of overload. A one-to-one mapping between queues and streams, with round-robin bitwise service of these queues, would eliminate this problem, but this algorithm, know as strict fair queueing, is expensive in terms of processing and space requirements. SFQ is a probabilistic variant of strict fair queueing. Instead of requiring that each flow have its own queue, SFQ has a fixed number of queues and uses a hashing function to map the IP source and destination address into one of the queues. Packets are entered into their assigned queues in a FIFO manner and are removed in a round-robin fashion between all nonempty queues (in the current implementation, the round-robin service is on a packet-by-packet basis). A seed to the hash function is occasionally perturbed, to allow a redistribution of the address pair mapping. This mapping redistribution is done to ensure that flows are not consistently mapped into the same queue, so that a well-behaved source is not penalized by an ill-behaved source if the flows happen to map to the same queue at some point in time. For a more complete description of SFQ, see the referenced paper by Paul E. McKenney.*

3 SFQ CONFIGURATION

SFQ has many parameters that can be tuned to improve its performance. These parameters include

- Individual queue depth
- Total number of queues
- Hash function
- Seed perturbation technique.

The setting of these parameters for these experiments was somewhat arbitrary, because the experiment design was on a small enough scale that these factors did not affect the outcome significantly in most cases. More research needs to be done to determine the optimum choice for larger scenarios. The parameters chosen for this set of experiments are described next.

The choice of the hash function is crucial to good behavior, because the hash function is responsible for distributing the packets among the queues. If the hash function is poor, many different flows map into the same queue and the behavior approaches that achieved with FIFO queueing. The current implementation provides five different hash functions using XOR or rotate operations. For the set of addresses used in the experiments, each hash function displayed similar behavior. We therefore chose an XOR type of hash function.

*McKenney, P.E. 1991. "Stochastic Fairness Queueing," in *Internetworking: Research and Experience*, Vol. 2, pp. 113-131.

The individual queue depth was set to 100. This parameter value was chosen to match the queue depth provided by the DARTnet kernel with its FIFO queueing discipline. Queue depth is a compile time constant and can be changed easily.

The total number of queues used is also an important factor in the behavior of SFQ. As mentioned earlier, too few queues result in behavior similar to FIFO; too many queues, and the behavior, and space overhead, resemble strict fair queueing if a suitable hash function is used. To ensure that the condition arises where more than one flow maps to the same queue during the lifetime of an experiment (i.e. a collision occurred during the hash computation), the number of queues was limited to 9. This number is controlled by a compile-time constant and is easy to change; however, in our implementation, the number of queues has to be equal to 2^m+1, where m is an integer. We are using a software implementation of the modulo operator, which works only for these values, because the current SPARC* architecture lacks the hardware support necessary for an efficient implementation. The modulo operator is used to reduce the result of the hashing function to the desired range.

As previously mentioned, collisions into the same queue will occur for the host addresses used in the experiments below. Thus, the experiments will show the results with and without seed perturbation. To preserve packet ordering, the current implementation increments the seed when all the queues are empty. In a congested router, it is anticipated that this will be infrequent; therefore, different techniques for seed perturbation need to be developed and tested.

4 EXPERIMENTS

The experiments were executed on DARTnet, a DARPA research testbed network. DARTnet is a cross-country T1 network† that connects research sites via T1 tail circuits. The routers and most of the hosts are SPARC 1+, with the exception of two hosts, MM6 and Malarky, which are SPARC 2s. Figure 1 illustrates the portion of the network used in these experiments.

The objectives of the experiments were to show the benefits and performance of SFQ. These experiments therefore demonstrate

- Fair utilization of available resources
- Starvation prevention
- Graceful degradation under overload conditions
- Resource usage.

Each experiment was run at least twice to verify its repeatability; however, in this report we will include only a single case. Each run of an experiment consisted of executing the experiment with a special kernel on the routers that support SFQ, with and without seed perturbation in most cases, and with the standard DARTnet kernel, which provided the FIFO queueing mechanism. The version number of the DARTnet kernel was 6. All traffic streams originated from machines using

*All product names mentioned in this report are the trademarks of their respective holders.

†Due to hardware constraints, the network operates at 1.344 Mb/s instead of 1.536 Mb/s.

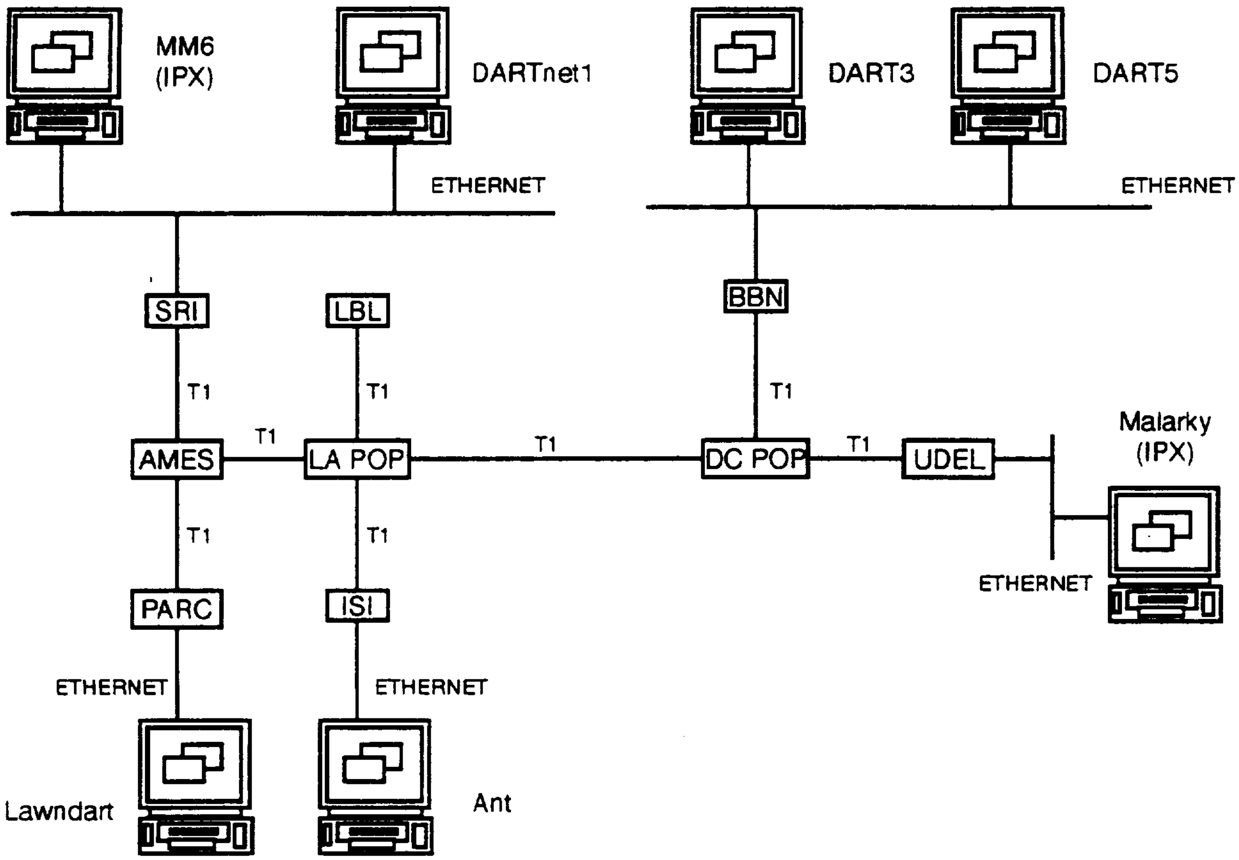


Figure 1. Experiment Topology

version 6 of the DARTnet kernel. Each stream in the experiment ran for 245 seconds, but the total time of the experiment was 335 seconds, due to the staggering of the start times for each stream. All experiments were designed to overload the link so that the network's behavior in times of congestion could be observed.

The DARTnet traffic generator (TG) was used to create the traffic streams. TG is an SRI-developed tool for creating high-quality and repeatable experiments on packet-switched networks. It executes as a source and sink program that enables experimenters to generate one-way traffic streams and gather statistical data about the transmission and reception of each stream. The TG is driven by a control language (script) that specifies different operating modes, protocols, addressing functions, traffic parameters, and execution times. At present, packet lengths and packet offer rates can be specified according to the following distributions: constant, uniform, exponential, and 2-state Markov. The delay and throughput for each of the experiments, therefore, is measured from user process to user process.

4.1 FAIR UTILIZATION

4.1.1 Objective and Procedure

This experiment was designed to show that streams receive equal portions of the available bandwidth. In this experiment, four equal UDP streams are created; these streams occupy 60 percent of the link capacity of a T1 line. The traffic distribution for each stream is identical: each stream's offer rate is exponentially distributed with a interarrival mean of 0.007692 seconds (~130 packets per second) and a constant size of 782 bytes (including UDP and IP headers). The source, destination, and direction for each stream is as follows (see Figure 2):

- Dartnet1 to Malarky
- Lawndart to Dart5
- LBL router to Dart3
- MM6 to Ant.

At the start of the experiment, the stream from DARTnet1 to Malarky and the stream from MM6 to Ant collide with the initial seed value.

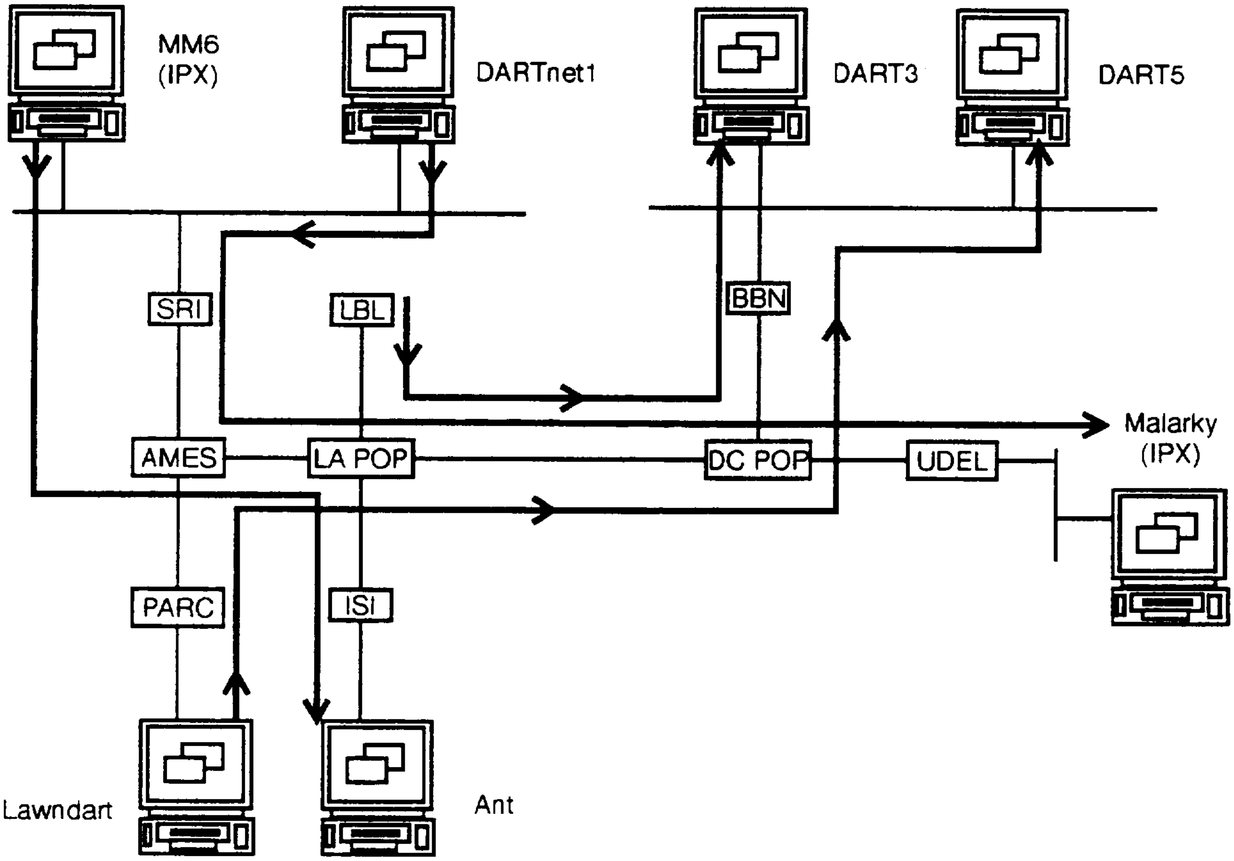


Figure 2. Fair Utilization Traffic Flow

4.1.2 Data Description

The graphs below show the offer rate, the SFQ throughput, and the FIFO throughput for each stream. The experiments were executed with and without seed perturbation: SFQ Experiment 29 ran without seed perturbation, and SFQ Experiment 49 ran with seed perturbation.

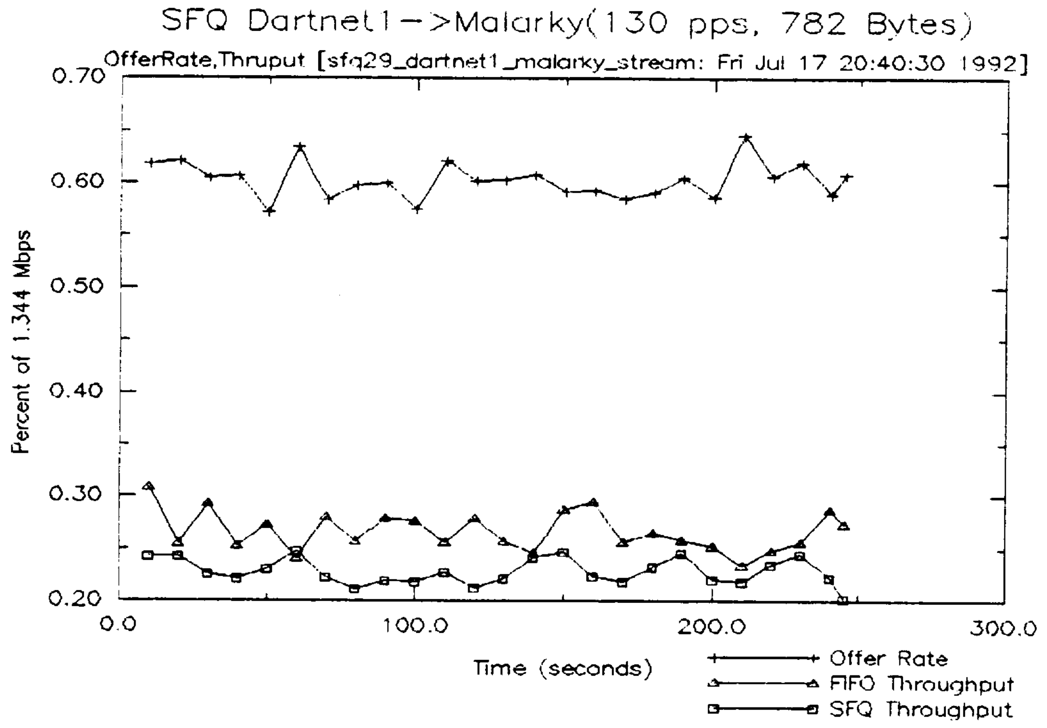
Table 1 summarizes the results of the experiments, including the average offer rate, average throughput, average delay, and delay variance for each stream.

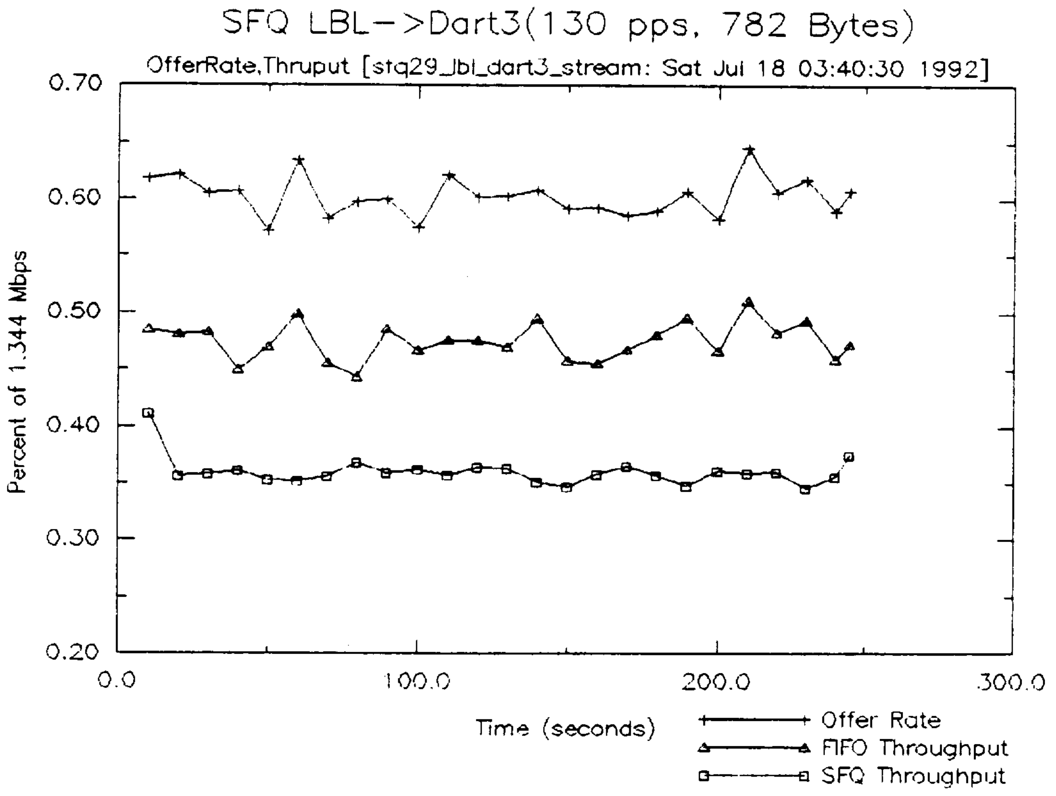
4.1.3 Results

The results of this experiment show that SFQ does provide better equal access to the available resources, while FIFO queueing allows the LBL to Dart3 to dominate the resource. Furthermore, seed perturbation does seems to improve SFQ performance when collisions occur. It is important to note that the throughput bottleneck occurs on the link from AMES to LA and on the link from LA POP to the DC POP. At each of these points in the network, three streams are trying to share the same link. Under perfect utilization, each stream would receive 32.3% of link capacity.* In experiment 49 with SFQ, the utilization for each stream was 31.8%, 31.6%, 31.7%, and 31.8%.

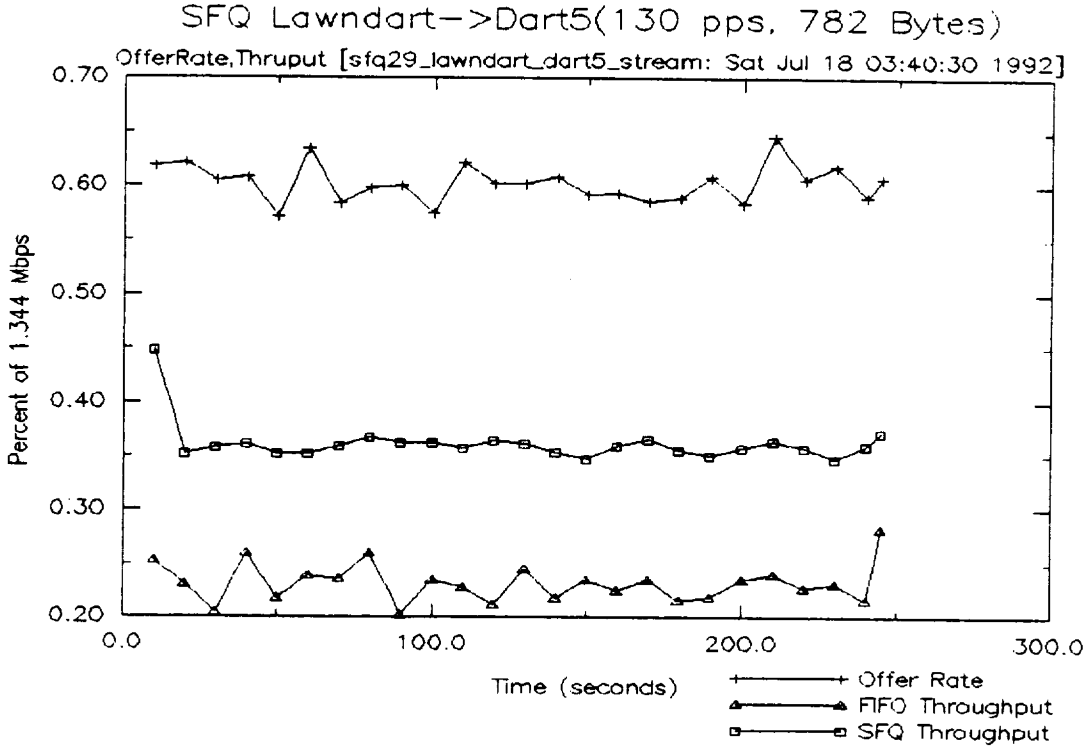
*In previous baseline measurements of the DARTnet FIFO kernel, the throughput for a packet size of 750 bytes was 96.9% of the link capacity.

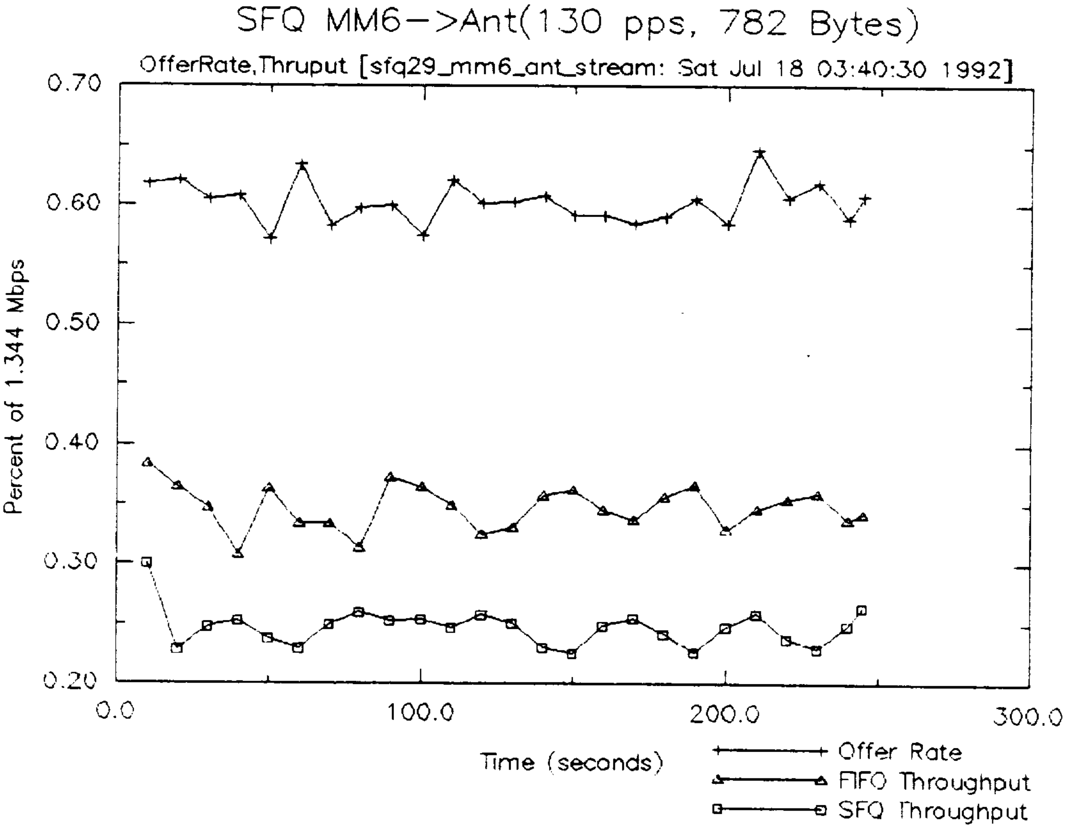
SFQ EXPERIMENT 29



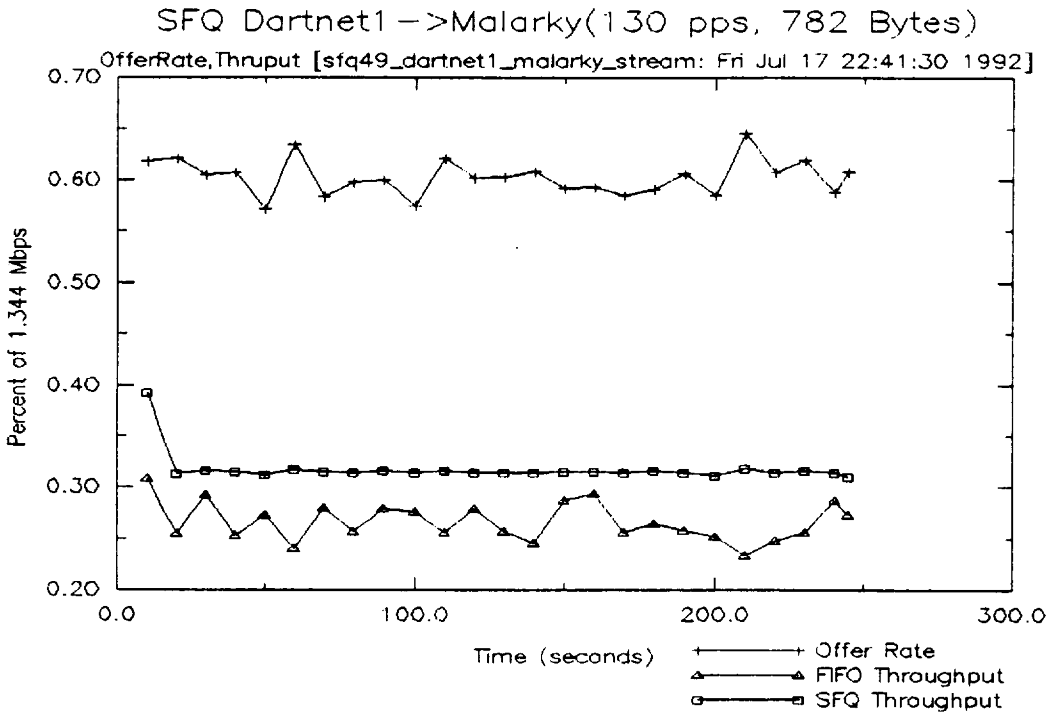


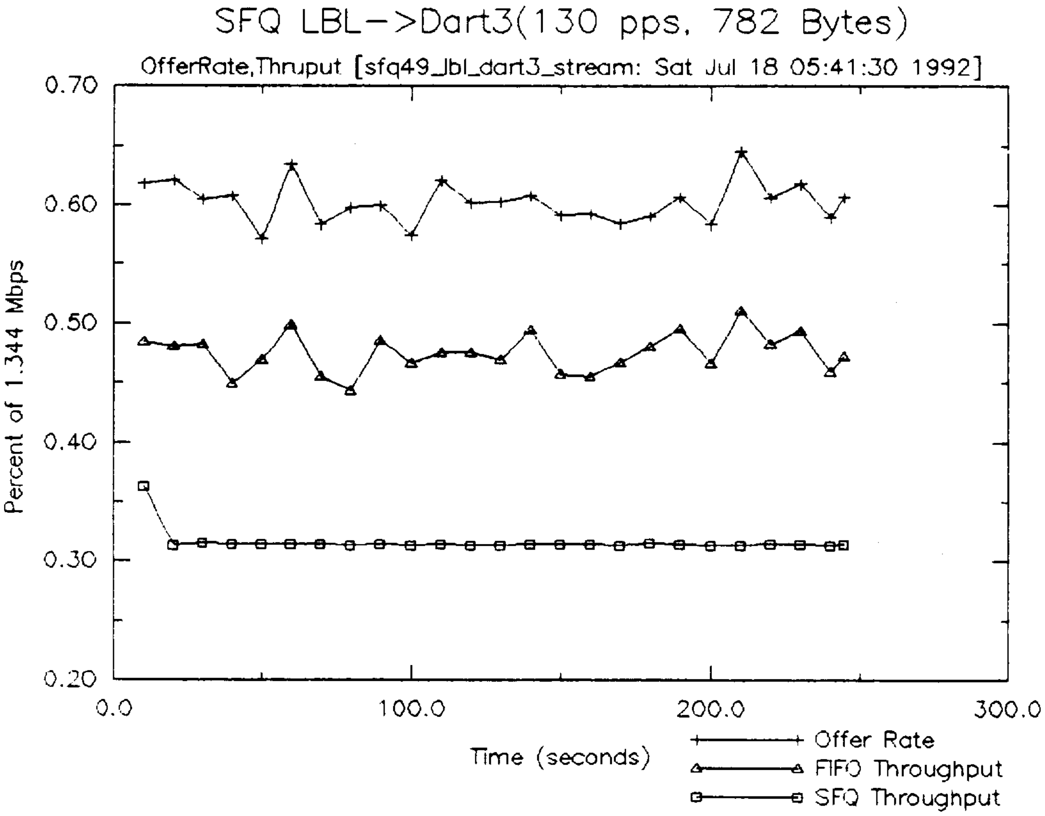
SFQ EXPERIMENT 29



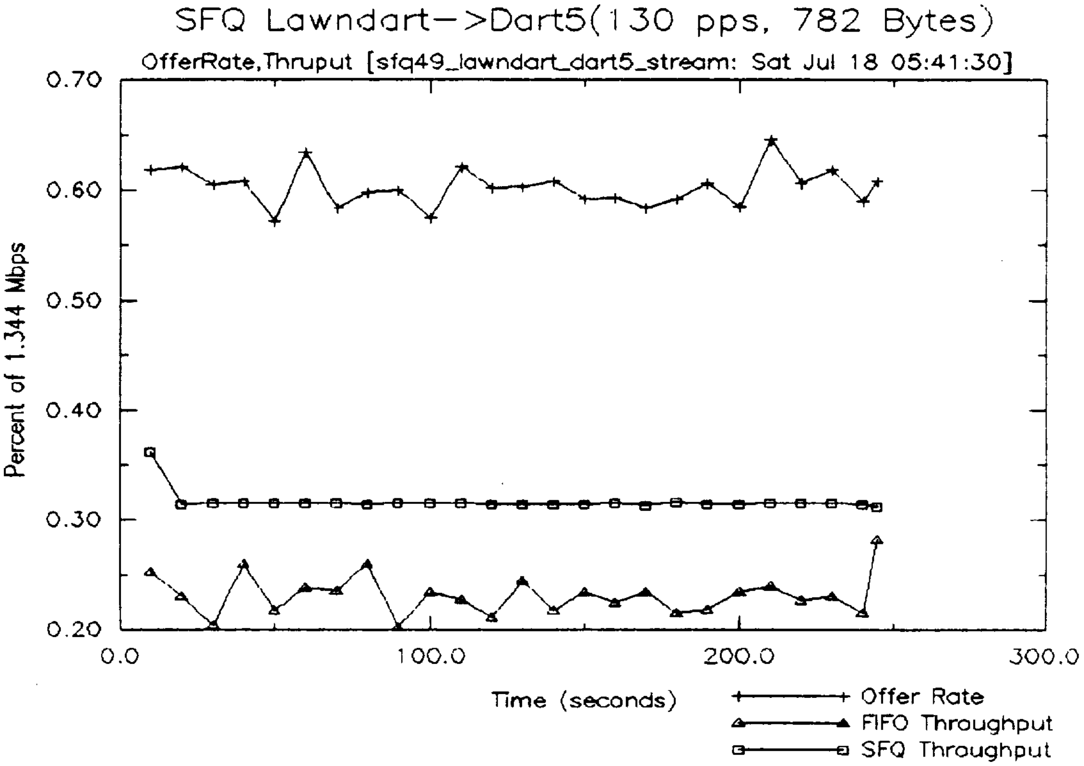


SFQ EXPERIMENT 49





SFQ EXPERIMENT 49



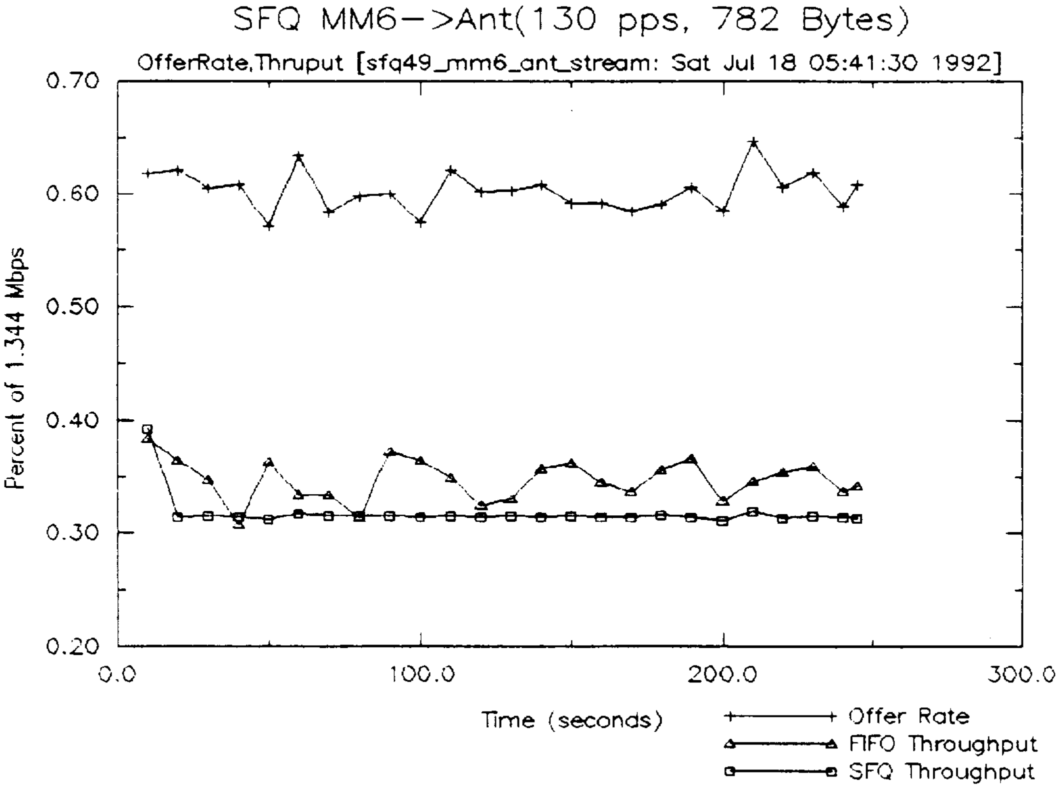


Table 1. Fair Utilization Results

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
Dartnet1 to Malarky	fifo29	60.26	26.64	1.4704	0.0057
	sfq29	60.25	22.85	1.5186	0.0074
	sfq49	60.25	31.84	2.8637	0.1215
LBL to Dart3	fifo29	60.25	47.49	0.5367	0.0007
	sfq29	60.26	36.00	1.3415	0.0095
	sfq49	60.26	31.62	1.5221	0.0095
Lawndart to Dart5	fifo29	60.25	23.08	1.0198	0.0029
	sfq29	60.25	36.15	2.2988	0.0384
	sfq49	60.25	31.72	1.9565	0.0849
MM6 to Ant	fifo29	60.26	34.73	0.9608	0.0024
	sfq29	60.25	24.71	1.4666	0.0071
	sfq49	60.25	31.84	2.4286	0.0398

4.2 STARVATION PREVENTION

4.2.1 Objective and Procedure

This experiment was designed to show SFQ's ability to prevent starvation. In the experiment, three unequal UDP streams are created. The source, destination, and direction of each stream are as follows (see Figure 3):

- Lawndart to Ant
- MM6 to Dart5
- Dartnet1 to Malarky.

The stream from Lawndart to Ant occupies 95% of the link capacity of a T1 line. This stream is exponentially distributed, with an interarrival mean of 0. 004902 seconds (~204 packets per second) and a constant packet size of 782 bytes (including UDP and IP headers). The streams from MM6 to Dart5 and Dartnet1 to Malarky each occupy 20% of the link capacity of a T1 line. Each stream is exponentially distributed, with an interarrival mean of 0.023256 seconds (~43 packets per second) and a constant packet size of 782 bytes. At the start of the experiment, the streams from Lawndart to Ant and from MM6 to Dart5 collide with the initial seed value.

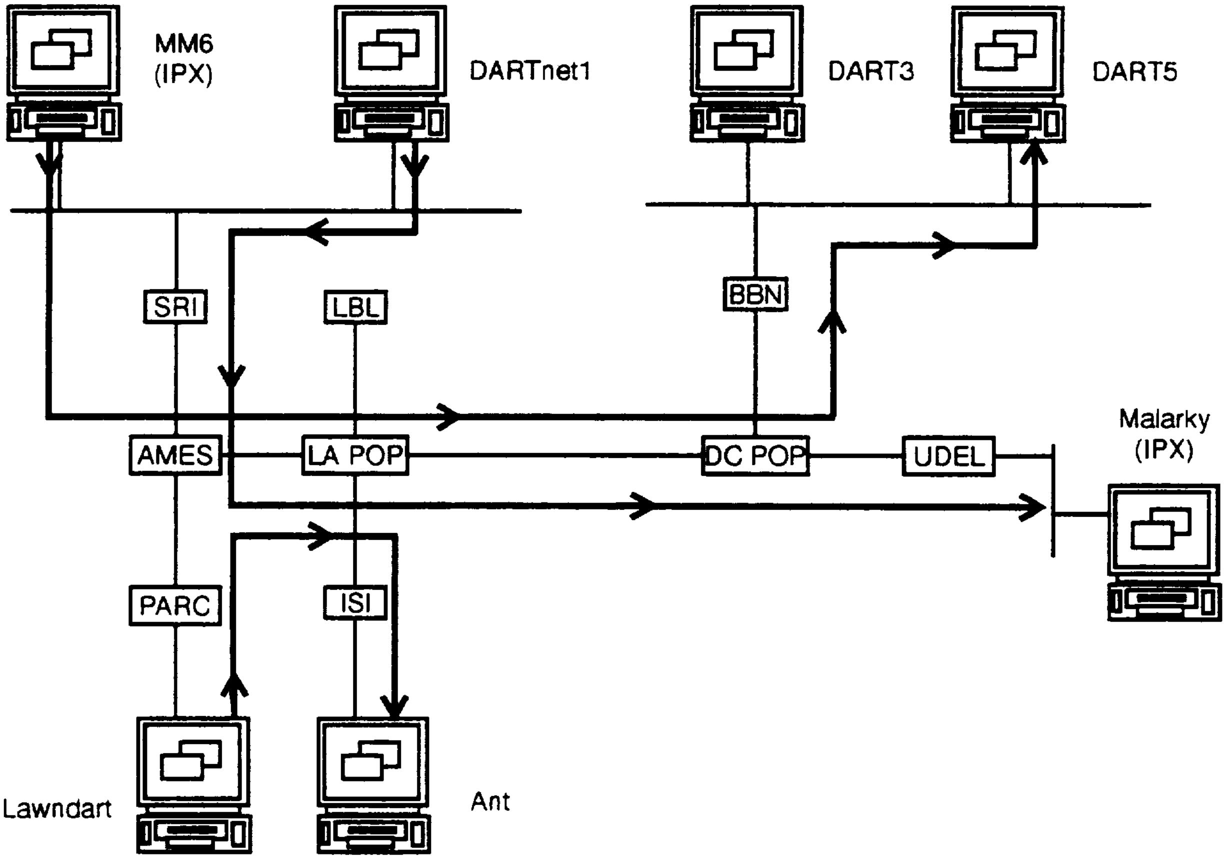


Figure 3. Starvation Prevention Traffic Flow

4.2.2 Data Description

Each graph below shows the offer rate, the SFQ throughput, and the FIFO throughput for each stream. The experiments were executed with and without seed perturbation: SFQ Experiment 28 ran without seed perturbation and SFQ Experiment 47 ran with seed perturbation.

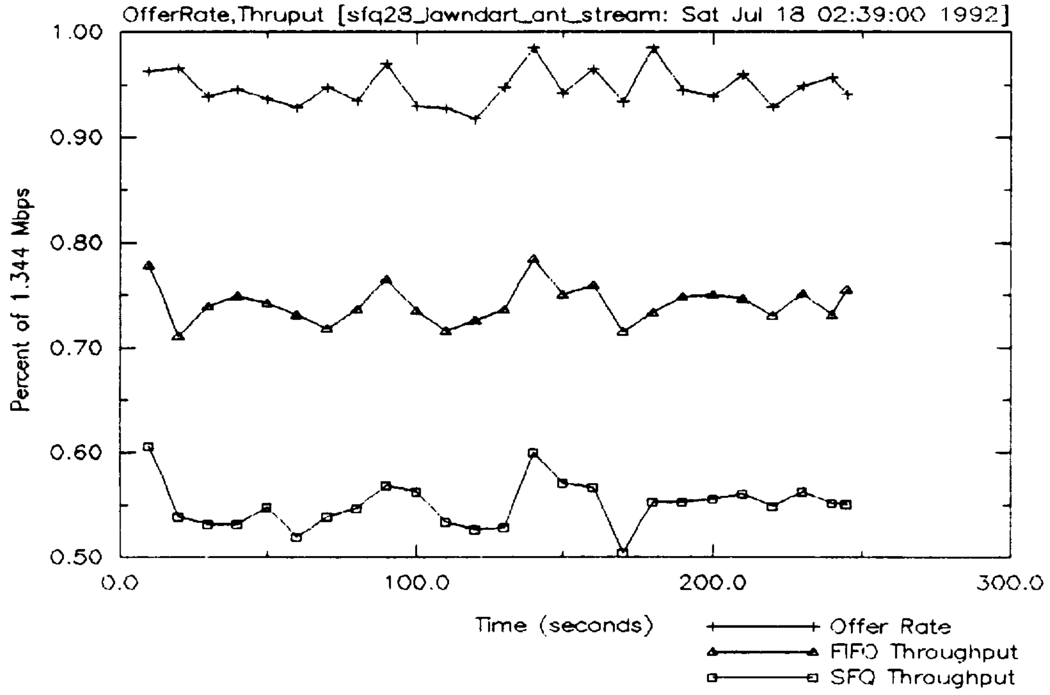
Table 2 summarizes the results of the experiments, including average offer rate, average throughput, average delay, and delay variance for each stream.

4.2.3 Results

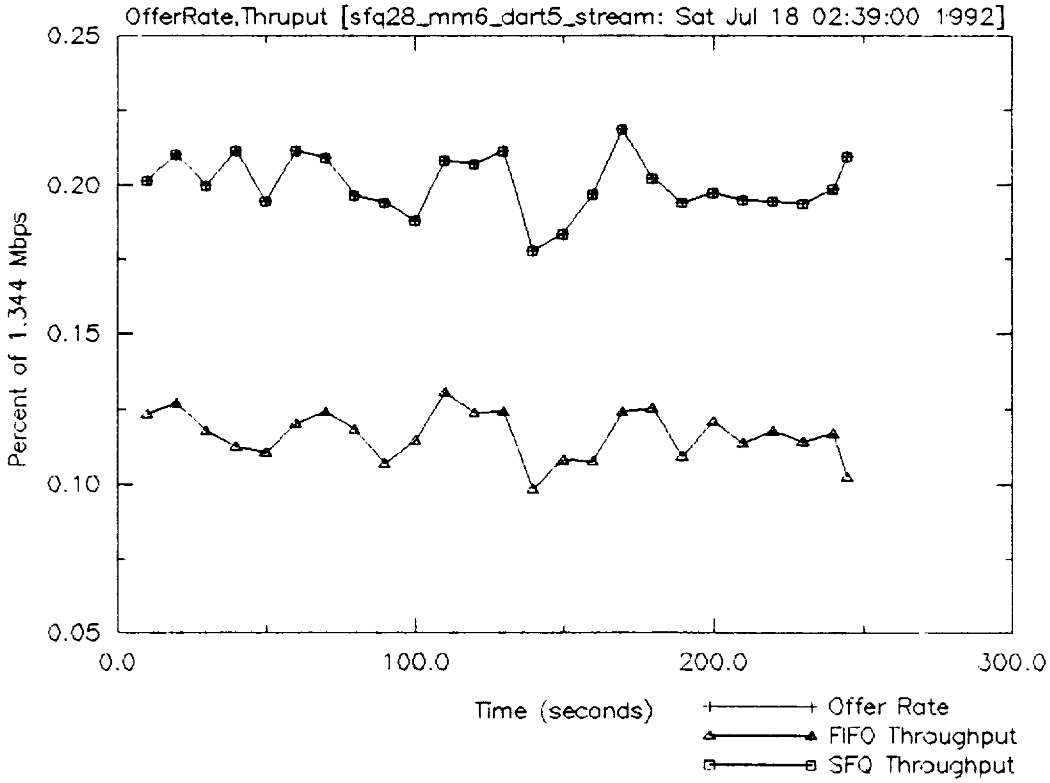
This experiment does show that SFQ helps prevent starvation by ensuring that those streams that demand less than their fair share of the available bandwidth receive their entire quota; while with FIFO queueing, they receive a disproportionate amount. In particular, with FIFO queueing, the stream that demanded ~95% of capacity received 74%, approximately 78% of its request. However, the two streams that wanted only 20% of the bandwidth received ~11%, or only 58% of their requested utilization. The variance in delay for the smaller streams was also less with SFQ than with FIFO: SFQ thus rewards "nice" users of the network.

SFQ EXPERIMENT 28

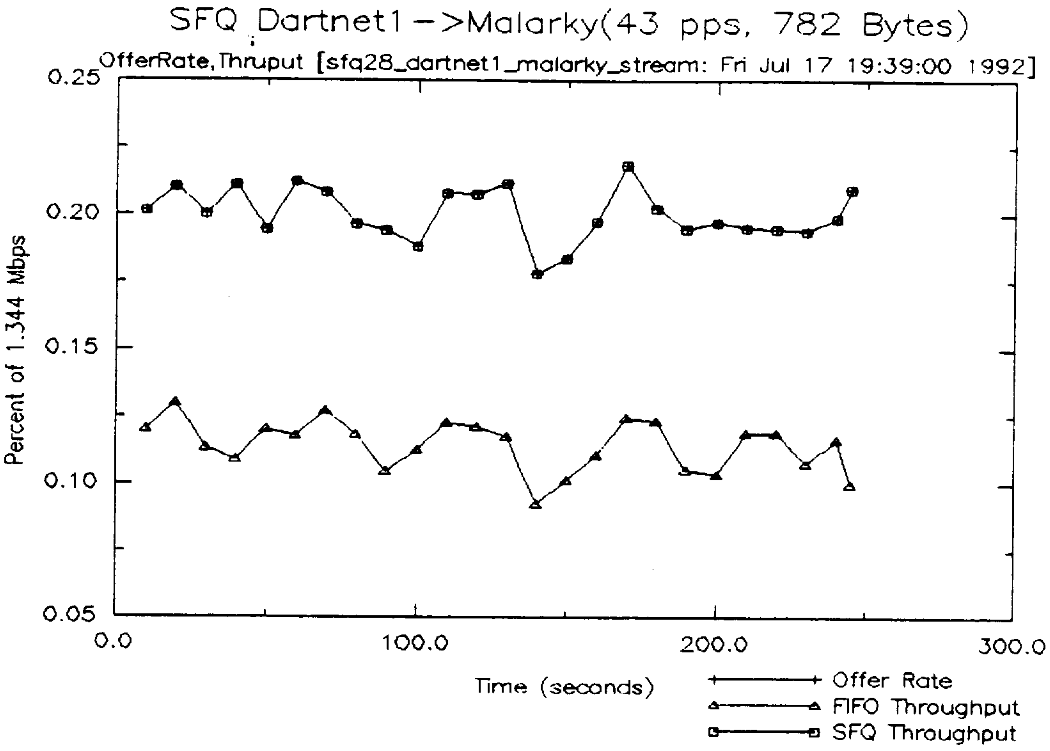
SFQ Lawndart−>Ant(204 pps, 782 Bytes)



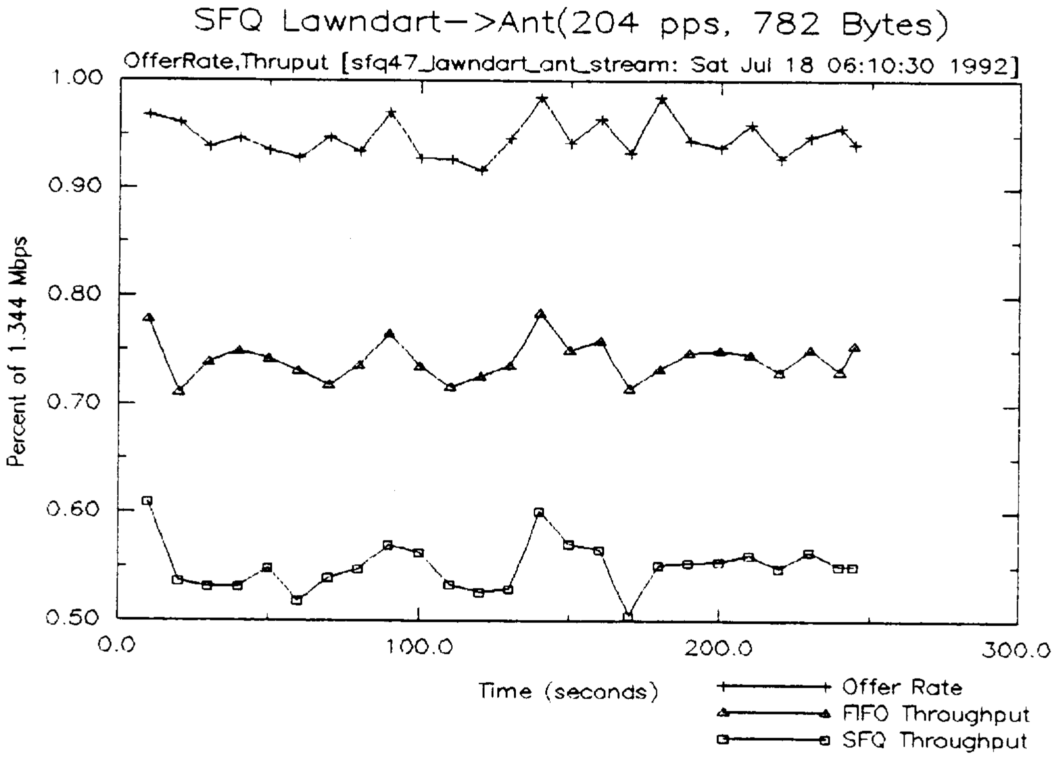
SFQ MM6−>Dart5(43 pps, 782 Bytes)



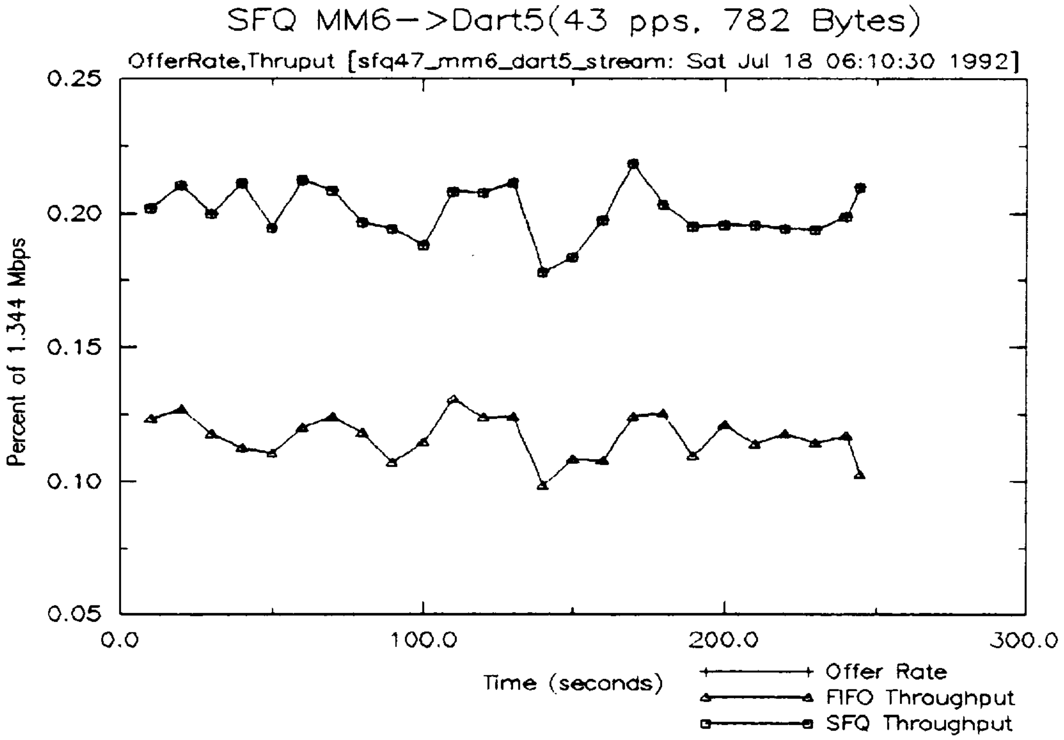
SFQ EXPERIMENT 28



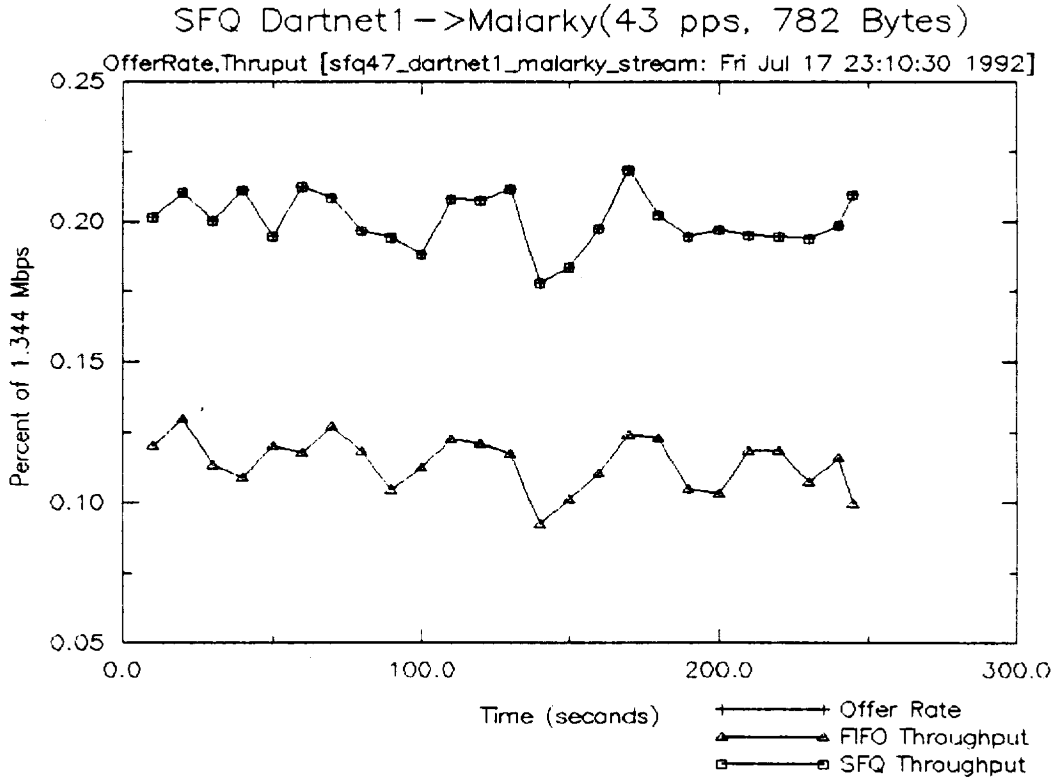
SFQ EXPERIMENT 47



SFQ EXPERIMENT 47


SFQ MM6->Dart5(43 pps, 782 Bytes)

OfferRate,Thruput [sfq47_mm6_dart5_stream: Sat Jul 18 06:10:30 1992]


SFQ Dartnet1->Malarky(43 pps, 782 Bytes)

OfferRate,Thruput [sfq47_dartnet1_malarky_stream: Fri Jul 17 23:10:30 1992]

Table 2. Starvation Prevention Results

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
Lawndart to Ant	fifo27	94.72	74.12	0.5727	0.0057
	sfq28	94.72	55.05	1.0888	0.0272
	sfq47	94.72	55.07	1.0897	0.0271
MM6 to Dart5	fifo27	20.01	11.68	0.5383	0.0007
	sfq28	20.01	20.01	0.0853	0.0004
	sfq47	20.01	20.01	0.0770	0.0003
Dartnet1 to Malarky	fifo27	20.01	11.44	0.5377	0.0008
	sfq28	20.01	20.01	0.0796	0.0003
	sfq47	20.01	20.01	0.0796	0.0004

4.3 GRACEFUL DEGRADATION

4.3.1 Objective and Procedure

This experiment was designed to show that SFQ degrades gracefully in periods of overload; each stream receives its fair share of the available bandwidth as more streams are added. In the experiment, four equal UDP streams were created; the streams occupy 60 percent of the link capacity of a T1 line. The traffic distribution for each stream is identical: each stream's offer rate is exponentially distributed, with a interarrival mean of 0.007692 seconds (~130 packets per second) and a constant size of 782 bytes (including UDP and IP headers). A stream is added every 30 seconds after the previous one. The source, destination, and direction of each stream, in the order in which the streams were created, is as follows (see Figure 4):

- Dartnet1 to Malarky
- LBL router to Dart3
- Lawndart to Dart5
- MM6 to Ant.

At the start of the experiment, the stream from DARTnet1 to Malarky and the stream from MM6 to Ant collide with the initial seed value.

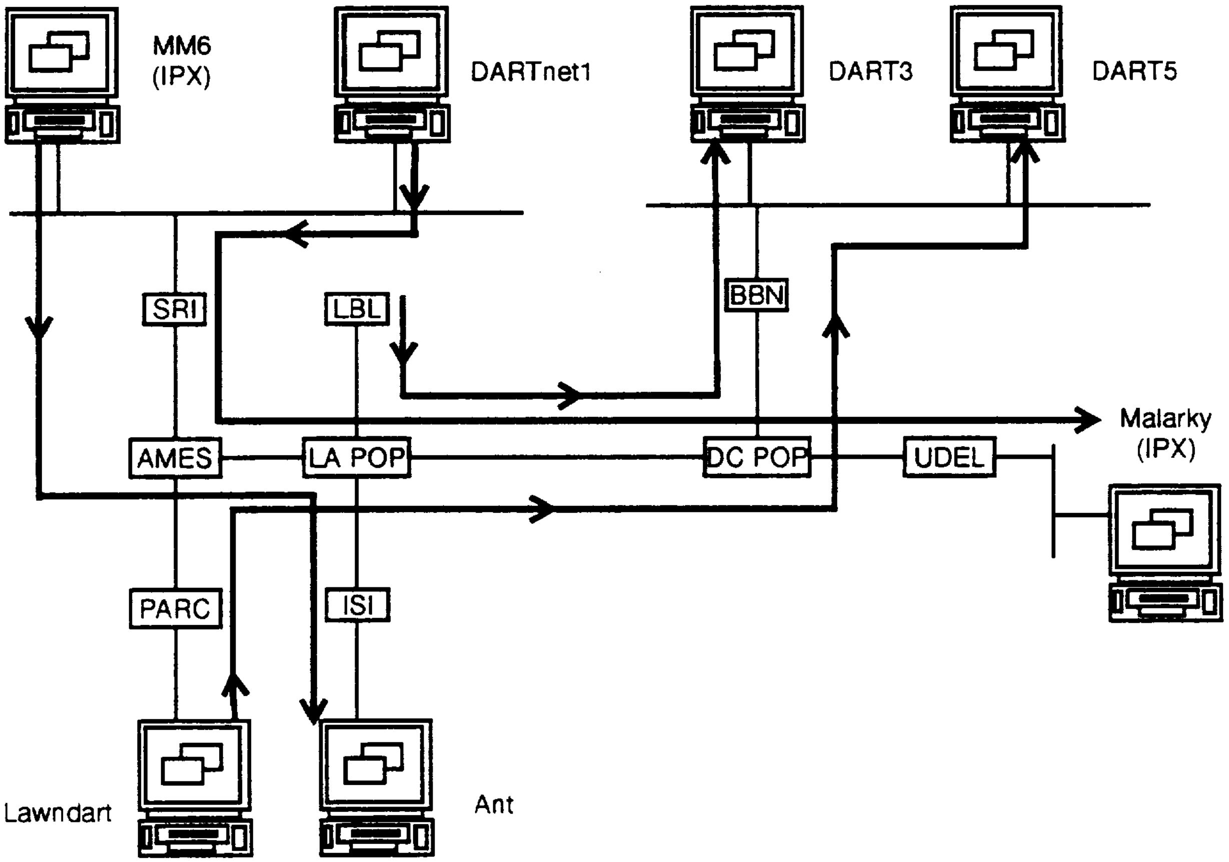


Figure 4. Graceful Degradation Traffic Flow

4.3.2 Data Description

The graphs below show the offer rate, the SFQ throughput, and the FIFO throughput for each stream. The experiments were executed with and without seed perturbation: SFQ Experiment 31 ran without seed perturbation and SFQ 41 ran with seed perturbation.

Tables 3 through 10 summarize the progression of the experiment, by including the average offer rate, average throughput, average delay, and delay variance for each significant time period for every stream. The time periods displayed are related to the times when streams are added and deleted, including a time period when all streams are executing simultaneously. To summarize,

- Table 3 shows the first 30 seconds, when only the Dartnet1 to Malarky stream exists.
- Table 4 shows the second 30-second interval, when the stream from LBL to Dart3 has been added.
- Table 5 shows the third 30-second interval, when the stream from Lawndart to Dart5 has been added.
- Table 6 shows the fourth 30-second interval, when the stream from MM6 to Ant has been added.
- Table 7 shows the next 125 seconds, when all the streams are running.
- Table 8 shows the next 30 seconds, when the stream from DARTnet to Malarky has dropped out and only the streams from LBL to Dart3, Lawndart to Dart5, and MM6 to Ant are running.
- Table 9 shows the next 30 seconds, when only the streams Lawndart to Dart 5 and from MM6 to Ant are present.
- Table 10 shows the final 30 seconds, when the stream from MM6 to Ant is the only one left.

As in the fair utilization experiment, the throughput bottleneck occurs on the links from AMES to LA and from LA POP to DC POP. The bottleneck on the AMES to LA link affects the streams from Dartnet1 to Malarky, Lawndart to Dart5, and MM6 to Ant; while the bottleneck on the link from LA POP to DC POP affects the streams from Dartnet1 to Malarky, LBL to Dart3, and Lawndart to Dart5. At each of these locations, the streams are trying to share the same link.

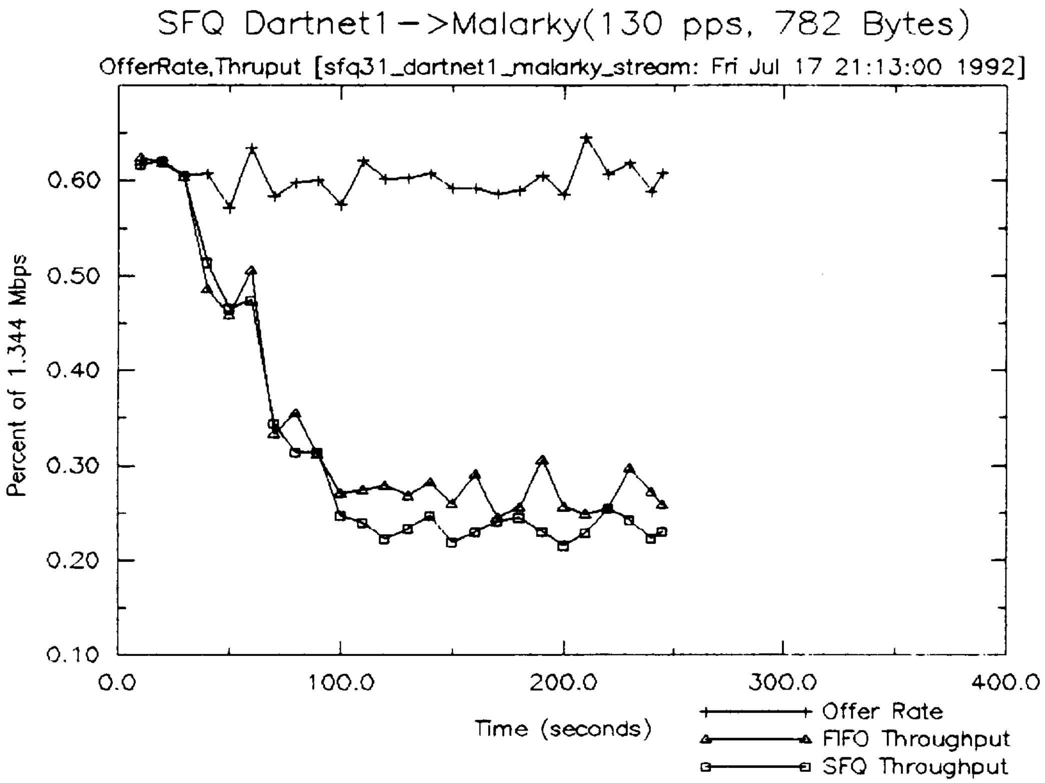
4.3.3 Results

The results of this experiment should be the following:

1. As the first three streams are added, the amount of throughput on each stream should decrease proportionately to the number of streams running, since they all use the LA POP to DC POP link.
2. As the fourth stream (MM6 to Ant) is added, there should be little effect on the first three streams, since the bottleneck has not substantially changed.
3. As Dartnet1 to Malarky drops out, the throughput on all the remaining streams should reach approximately 50%.
4. As LBL to Dart3 drops out, there should be no real effect on the two remaining streams, because the AMES-LA bottleneck now comes into play.
5. As the Lawndart to Dart5 stream drops out, throughput on the stream from MM6 to Ant should match the offer rate.

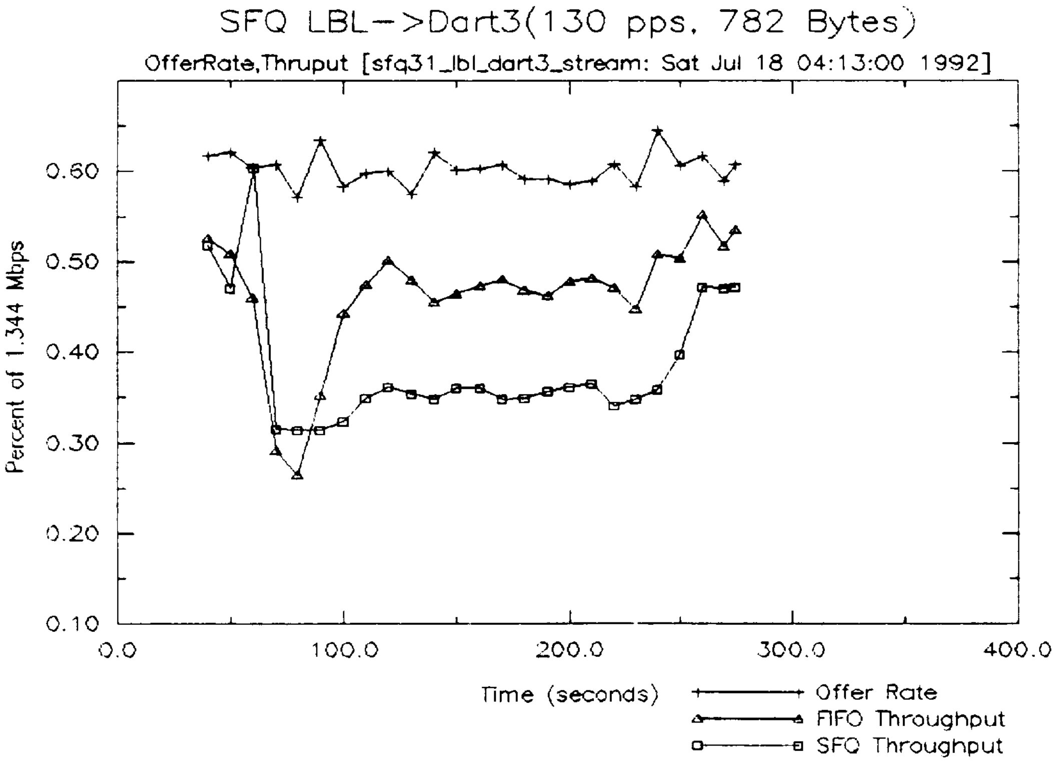
In general, both SFQ and FIFO exhibited the behavior described above. However, SFQ seems to be "more" fair overall than FIFO queueing. The range of throughputs achieved under SFQ more closely matched the ideal; under FIFO queueing there may be a tendency for a stream to "dominate" the available resources under heavy utilization (see Table 7). Not unexpectedly, "strict" fair queueing (SFQ with no collisions) behaved better than SFQ with seed perturbation (see Table 5). However, in one instance, FIFO queueing behaved better than SFQ with seed perturbation (see Table 5). This underlines the need to find optimal hashing functions and good seed perturbation techniques.

SFQ EXPERIMENT 31



SFQ Dartnet1->Malarky(130 pps, 782 Bytes)

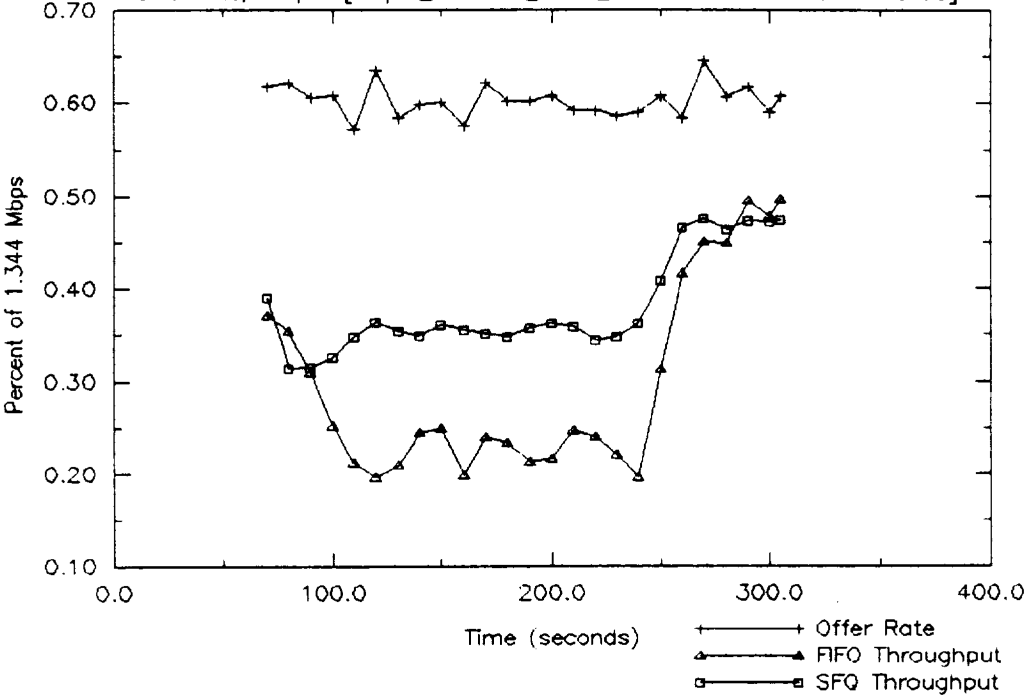
OfferRate,Thruput [sfq31_dartnet1_malarky_stream: Fri Jul 17 21:13:00 1992]



SFQ LBL->Dart3(130 pps, 782 Bytes)

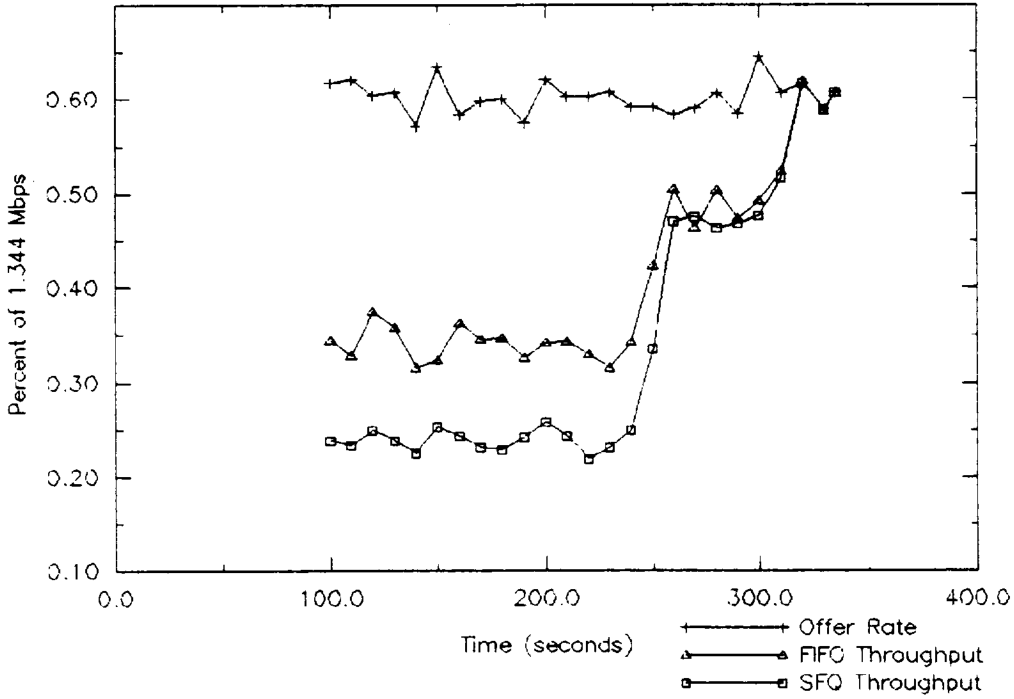
OfferRate,Thruput [sfq31_lbl_dart3_stream: Sat Jul 18 04:13:00 1992]

SFQ EXPERIMENT 31



SFQ Lawndart->Dart5(130 pps, 782 Bytes)

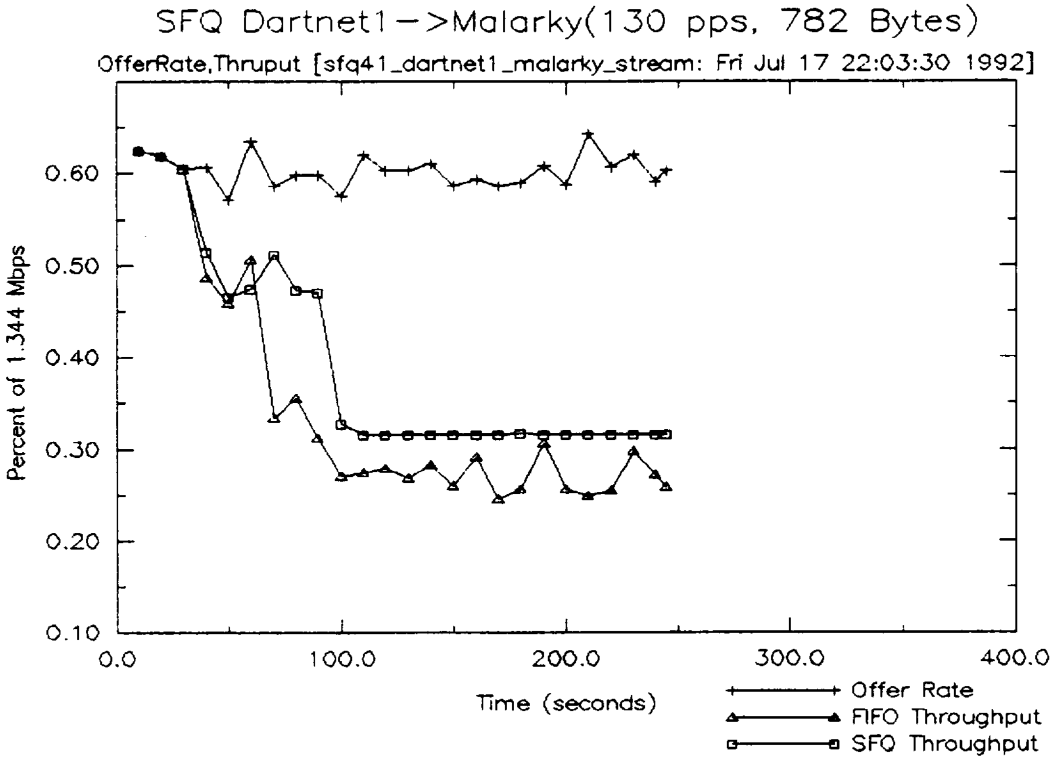
OfferRate,Thruput [sfq31_lawndart_dart5_stream: Sat Jul 18 04:13:00]

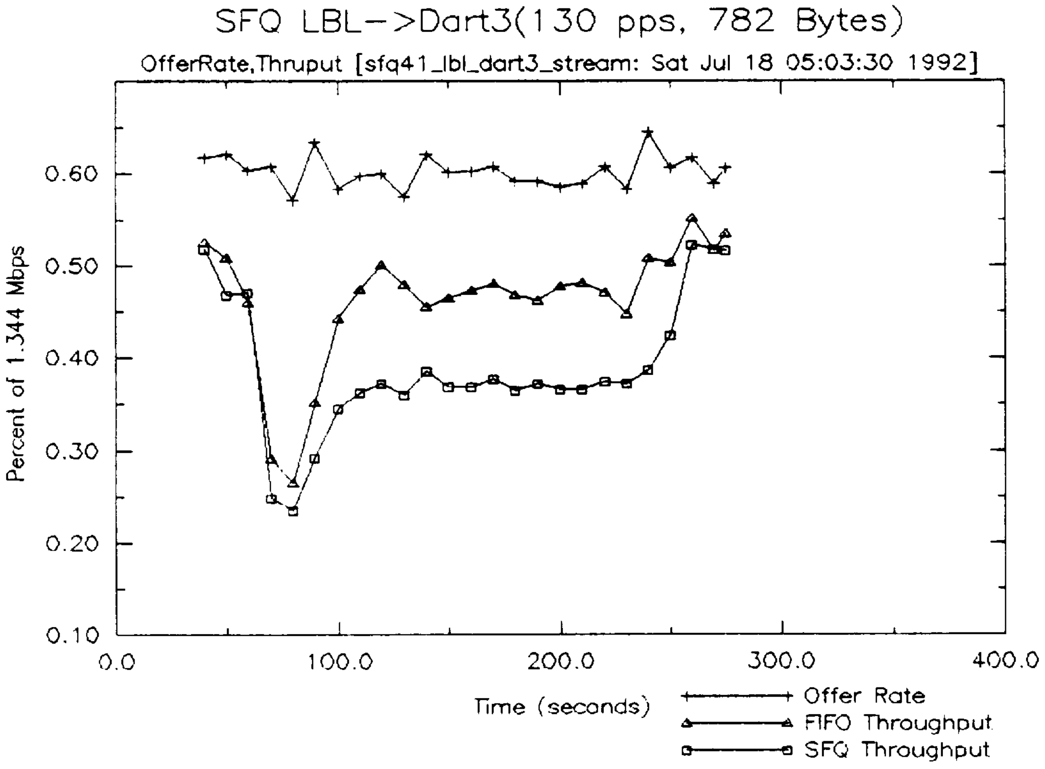


SFQ MM6->Ant(130 pps, 782 Bytes)

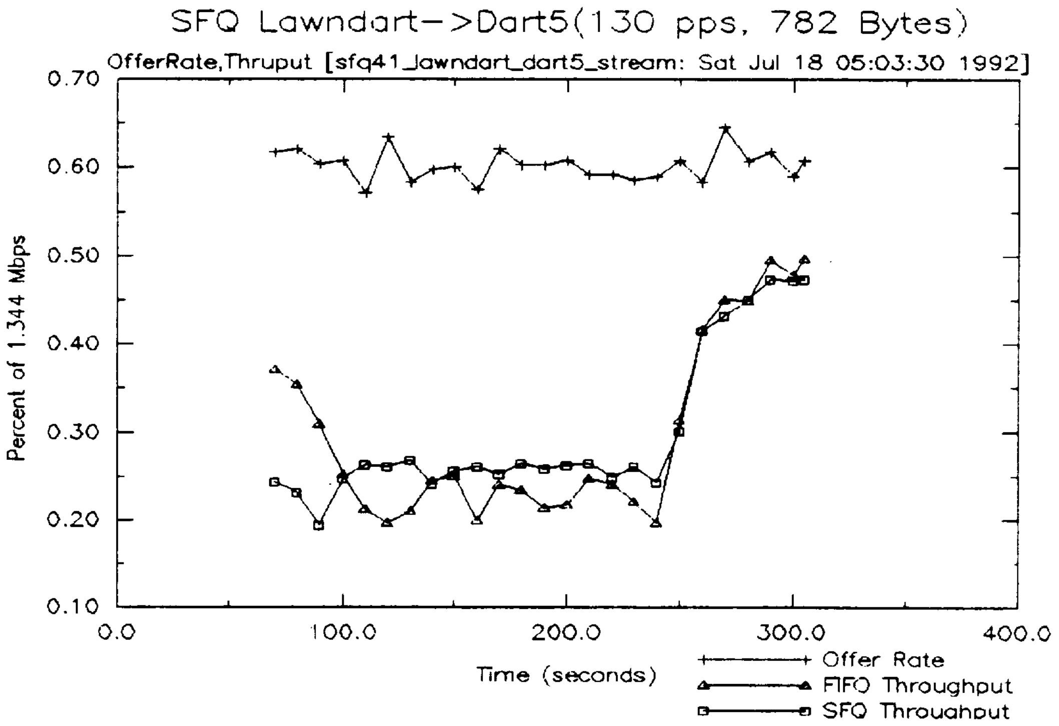
OfferRate,Thruput [sfq31_mm6_ant_stream: Sat Jul 18 04:13:00 1992]

SFQ EXPERIMENT 41





SFQ EXPERIMENT 41



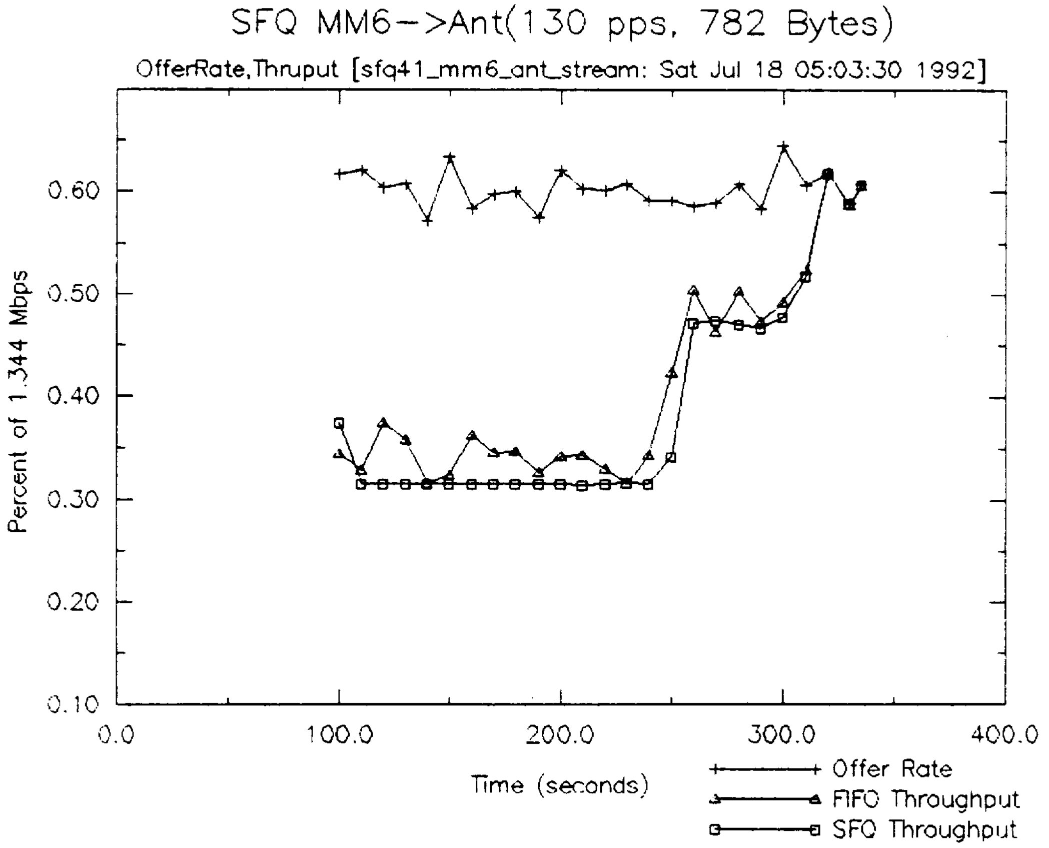


Table 3. Graceful Degradation Results From Time 0 to 30 Seconds*

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
Dartnet1 to Malarky	fifo31	61.61	61.61	0.0605	0.000044
	sfq31	61.47	61.47	0.0583	0.000049
	sfq41	61.63	61.63	0.0588	0.000051

*Time is approximate.

Table 4. Graceful Degradation Results From Time 30 to 60 Seconds*

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
Dartnet1 to Malarky	fifo31	60.39	48.35	0.5212	0.00503
	sfq31	60.43	48.53	0.9591	0.04639
	sfq41	60.39	48.47	0.9537	0.04045
LBL to Dart3	fifo31	61.45	49.79	0.5179	0.00557
	sfq31	61.45	48.58	0.9650	0.04815
	sfq41	61.46	48.56	0.9605	0.0429

*Time is approximate.

Table 5. Graceful Degradation Results From Time 60 to 90 Seconds*

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
Dartnet1 to Malarky	fifo31	59.38	33.34	0.99504	0.00743
	sfq31	59.38	32.43	2.4168	0.05466
	sfq41	59.39	48.52	1.8293	0.05488
LBL to Dart3	fifo31	60.45	30.24	0.5426	0.00004
	sfq31	60.45	31.44	1.5327	0.00016
	sfq41	60.45	25.85	1.0463	0.00005
Lawndart to Dart5	fifo31	61.47	34.43	0.9950	0.00896
	sfq31	61.47	33.98	2.3087	0.26848
	sfq41	61.47	22.31	1.9382	0.05580

*Time is approximate.

Table 6. Graceful Degradation Results From Time 90 to 120 Seconds*

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
Dartnet1 to Malarky	fifo31	59.97	27.42	1.4656	0.00532
	sfq31	59.95	23.69	1.6404	0.06492
	sfq41	59.95	31.90	1.9922	0.00157
LBL to Dart3	fifo31	59.38	47.29	0.5370	0.00012
	sfq31	59.38	34.47	1.3963	0.00701
	sfq41	59.38	36.03	0.8097	0.00358
Lawndart to Dart5	fifo31	60.43	22.02	1.0237	0.00010
	sfq31	60.43	34.62	2.2364	0.00730
	sfq41	60.43	25.71	2.2667	0.00098
MM6 to Ant	fifo31	61.47	34.96	0.9547	0.00582
	sfq31	61.46	24.07	1.4622	0.00585
	sfq41	61.46	33.50	1.8455	0.13713

*Time is approximate.

Table 7. Graceful Degradation Results From Time 120 to 245 Seconds*

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
Dartnet1 to Malarky	fifo31	60.25	26.94	1.4840	0.00021
	sfq31	60.20	23.42	1.5336	0.00026
	sfq41	60.21	31.54	2.0037	0.00025
LBL to Dart3	fifo31	60.19	47.28	0.5370	0.00013
	sfq31	60.20	35.46	1.3625	0.00223
	sfq41	60.19	37.23	0.7953	0.00008
Lawndart to Dart5	fifo31	59.66	22.62	1.0238	0.00012
	sfq31	59.62	35.48	2.3359	0.00245
	sfq41	59.66	22.62	1.0238	0.00012
MM6 to Ant	fifo31	60.04	33.93	0.9735	0.00012
	sfq31	60.03	23.91	1.4822	0.00010
	sfq41	60.01	31.53	1.9654	0.00024

*Time is approximate.

Table 8. Graceful Degradation Results From Time 245 to 275 Seconds*

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
LBL to Dart3	fifo31	60.01	53.26	0.5177	0.00109
	sfq31	60.02	46.52	1.0397	0.00238
	sfq41	60.01	51.16	0.5553	0.00207
Lawndart to Dart5	fifo31	61.29	42.86	0.9837	0.00778
	sfq31	61.83	46.63	2.0102	0.00273
	sfq41	61.79	41.23	1.5244	0.01237
MM6 to Ant	fifo31	58.70	48.47	0.5099	0.00276
	sfq31	58.85	46.30	1.0008	0.00304
	sfq41	58.85	45.49	1.0091	0.01186

*Time is approximate.

Table 9. Graceful Degradation Results From Time 275 to 305 Seconds*

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
Lawndart to Dart5	fifo31	60.01	43.82	0.5380	0.00029
	sfq31	60.01	47.23	1.0401	0.002752
	sfq41	60.01	47.30	1.0317	0.000472
MM6 to Ant	fifo31	61.79	48.68	0.5028	0.00016
	sfq31	61.78	47.17	0.9937	0.00070
	sfq41	61.77	47.22	0.9941	0.00075

*Time is approximate.

Table 10. Graceful Degradation Results From Time 305 to 335 Seconds*

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
MM6 to Ant	fifo31	60.01	60.01	0.0313	0.001457
	sfq31	60.01	59.78	0.0551	0.01628
	sfq41	60.01	59.81	0.0567	0.01707

*Time is approximate.

4.4 RESOURCE USAGE

4.4.1 Objective and Procedure

SFQ Experiment 24 experiment is similar in design to the experiment on starvation prevention (see Subsection 4.2). However, instead of three UDP streams flowing in the same direction, this experiment used two UDP streams flowing in one direction, and another stream in the reverse direction. This was to done to test SFQ's performance in relation to FIFO queueing. It also provided a stress test of the network, since previous experiments with cross streams failed (blackouts occurred).The source, destination, and direction of each stream are as follows (see Figure 5):

- Lawndart to Ant
- MM6 to Dart5
- Malarky to Dartnet1.

The streams from Lawndart to Ant and Malarky to Dartnet1 occupy 95% of the link capacity of a T1 line. Each of these streams is exponentially distributed, with an interarrival mean of 0.004902 seconds (~204 packets per second) and a constant packet size of 782 bytes (including UDP and IP

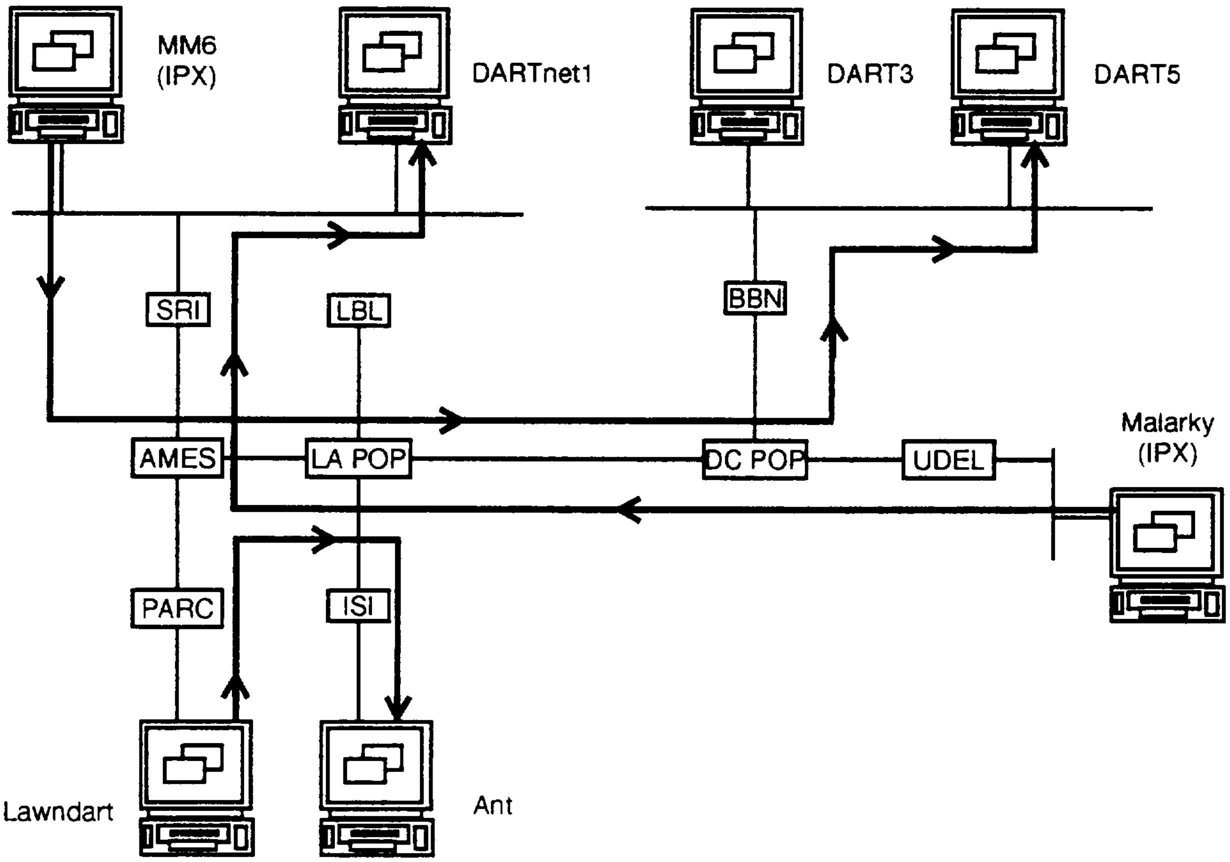


Figure 5. Resource Usage Traffic Flow

headers). The stream from MM6 to Dart5 occupies 20% of the link capacity of a T1 line. This stream is exponentially distributed, with an interarrival mean of 0.023256 seconds (~43 packets per second) and a constant packet size of 782 bytes.

4.4.2 Data Description

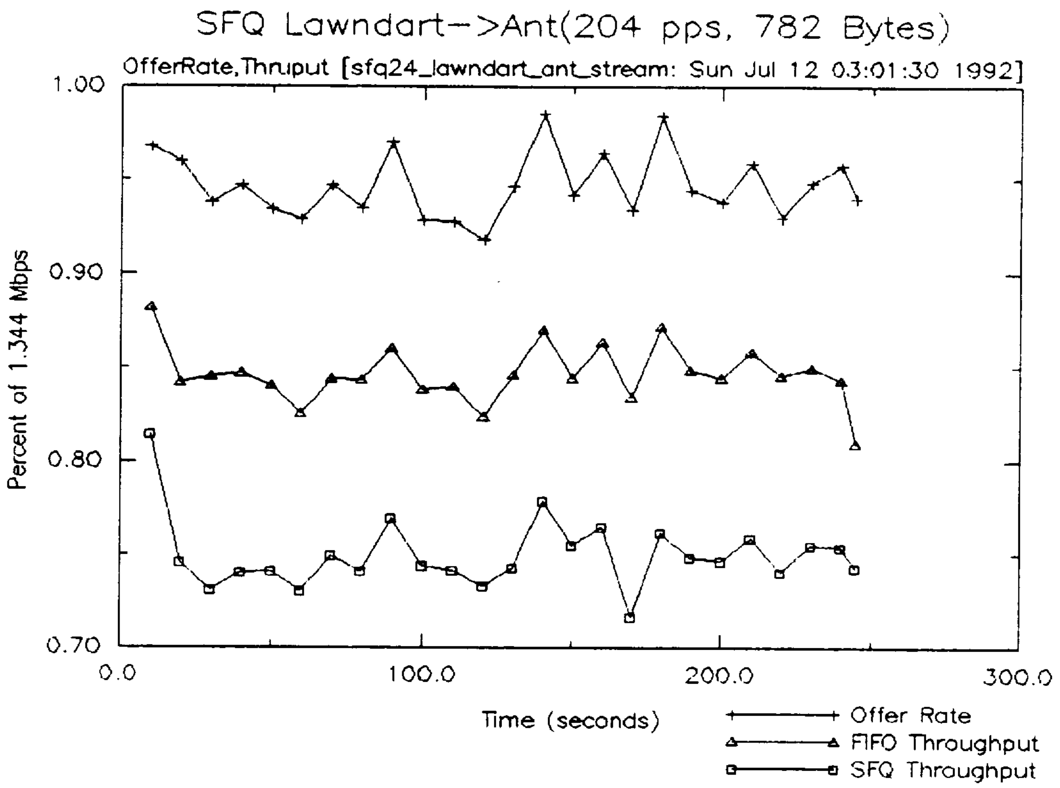
Each graph below shows the offer rate, the SFQ throughput, and the FIFO throughput for each stream. The experiment was executed without seed perturbation, which was not of interest.

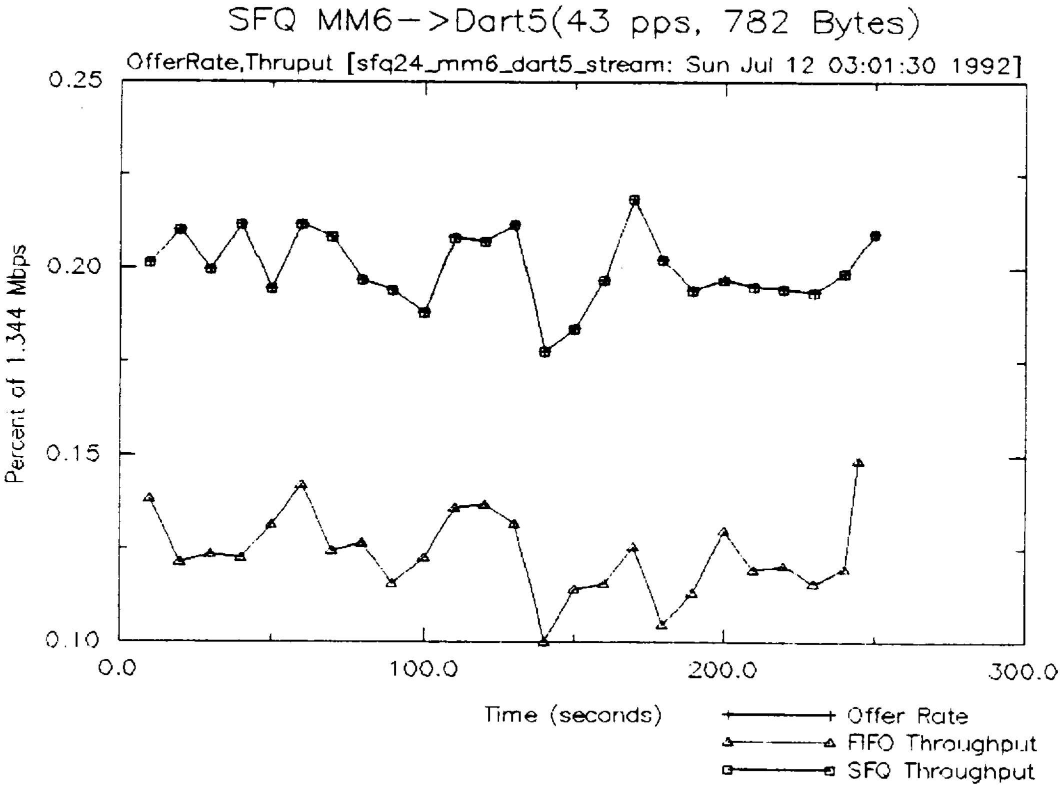
Table 11 summarizes the results of the experiments, including average offer rate, average throughput, average delay, and delay variance for each stream.

4.4.3 Results

This experiment does show that SFQ does not significantly affect the throughput of the network, compared with FIFO queueing. At ~95% offer load, the average throughput for FIFO queueing was 94.68%, while for SFQ it was 94.46%. As in all experiments, the average delay and variance in delay is greater with SFQ than with FIFO, because SFQ requires more processing. However, when a stream is competing for scarce resources, the variance seems to be lower with SFQ for streams that are demanding less than their share.

SFQ EXPERIMENT 24





SFQ EXPERIMENT 24

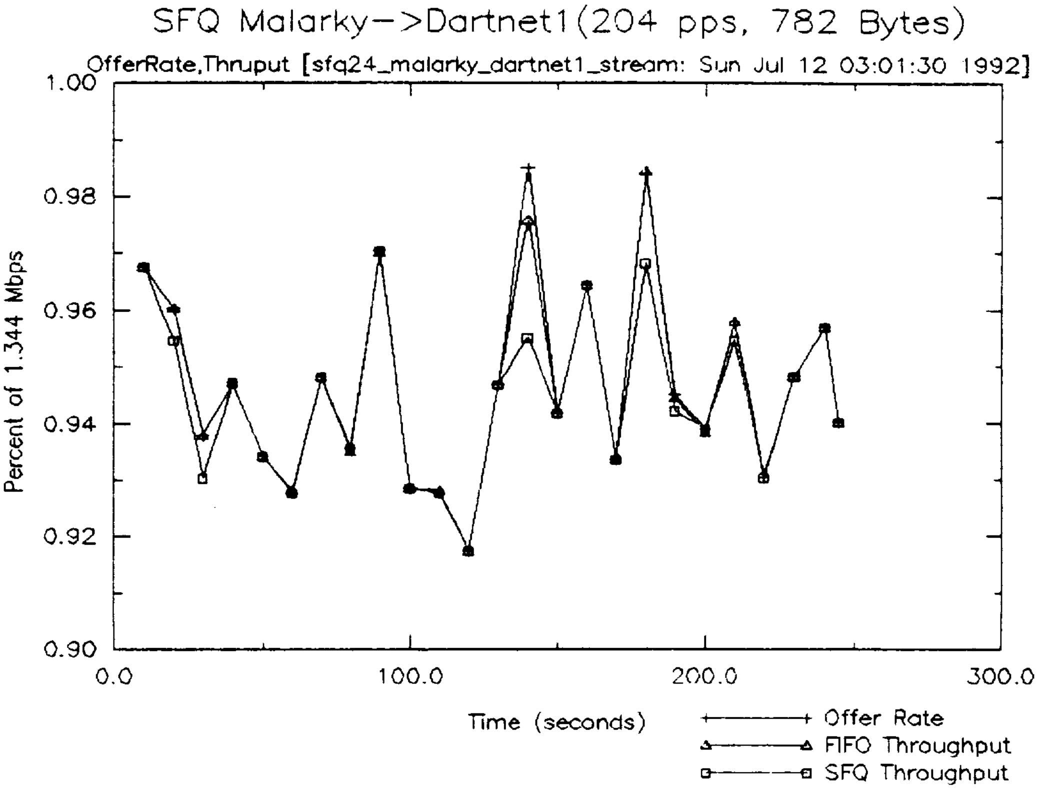


Table 11. Resource Usage

STREAM	EXPERIMENT NUMBER	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (seconds)	DELAY VARIANCE
Lawndart to Ant	fifo21	94.72	84.71	0.5714	0.0065
	sfq24	94.72	74.99	0.8562	0.0189
MM6 to Dart5	fifo21	20.01	12.34	0.5337	0.0015
	sfq24	20.01	20.01	0.0802	0.0001
Malarky to Dartnet1	fifo21	94.72	94.68	0.1455	0.0069
	sfq24	94.72	94.46	0.2808	0.0153

5 CONCLUSIONS

SFQ is an efficient queueing discipline for providing equal access to the available bandwidth. The isolation of the streams helps to ensure that no stream receives more than its fair share and that each stream degrades gracefully as more streams are added. SFQ also seems to possess very good scaling properties; but more work needs to be done to verify this. In particular, the choice of hash function and seed perturbation technique needs further investigation. The current choices may prove inadequate in a more stressful environment.

Appendix E

REPORT ON SFQ AND VIRTUALCLOCK: A HYBRID ALGORITHM

Technical Report • October 1993

A HYBRID ALGORITHM FOR COMBINING BEST-EFFORT AND RESOURCE RESERVATION SERVICE

Barbara A. Denny, Computer Scientist
Information and Telecommunications Sciences Center

Project 8600
ITAD-8600-TR-93-168R

Prepared for:

National Aeronautics and Space Administration (NASA)
Ames Research Center
Moffett Field, California 94035

Attn: Dr. Henry Lum, Code RI, M/S:244-7

and

Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, Virginia 22203-1714

Attn: Dr. Paul Mockapetris

Approved by:

Boyd C. Fair, Director
Information and Telecommunications Sciences Center

Michael S. Frankel, Vice President and Director
Information, Telecommunications, and Automation Division

CONTENTS

FIGURES

TABLES

ACKNOWLEDGEMENTS

We wish to thank Charles Lynn of Bolt Beranek and Newman Inc. for providing us with the ST-II software, including VirtualClock and the traffic control abstraction software. His quick resolution of some implementation issues was an immense help, enabling us to continue our work in a timely manner.

EXECUTIVE SUMMARY

SRI International (SRI) has developed an improved queueing algorithm, known as Stochastic Fairness Queueing (SFQ), for best-effort traffic (i.e., traffic that does not require any guaranteed service). SFQ is a probabilistic variant of strict fair queueing: instead of a single queue being allocated per flow, a fixed number of queues is used and a hash function maps the IP source and destination to a particular queue. However, new applications, such as audio and video, require certain delivery constraints to be met that SFQ was not designed to provide. Many new scheduling algorithms are being developed to meet these resource needs. A hybrid algorithm combining the strengths of SFQ with a resource reservation algorithm would better meet all the needs of the users of a network.

This report covers the design, implementation, and experimentation with such an algorithm. VirtualClock was selected as the resource reservation to combine with SFQ, because other algorithms and approaches were not mature enough or not in a form available for our use. VirtualClock is a rate-based scheduling algorithm that orders packets in a single queue according to a timestamp. This timestamp is based on the average arrival rate specified in the reservation request. For the routers to receive this resource request, a mechanism must exist to pass the flow specification. ST-II provides this functionality, and since a VirtualClock implementation was included in the release of the ST-II protocol, we are using ST-II. To implement our new algorithm, we chose to keep each queue structure and algorithm as close to the original design as possible, and only modify the procedure to select which of the packets to transmit next. We investigated two approaches. The first was a simple priority mechanism that always gives VirtualClock packets priority over SFQ packets. The second approach was to interleave the packets from the two queues by reinterpreting the timestamp used to order the VirtualClock packets as a time to send. If it is time to transmit, or past time to transmit, the packet is taken from the VirtualClock queue; otherwise, a packet from SFQ is taken. If this condition cannot be met, the algorithm always tries to find a packet to send. (See Subsection 2.4 for details).

The results of the experiment we performed do show that one can effectively combine SFQ with VirtualClock to achieve the benefit of SFQ (equal access to the available bandwidth), while preserving the average rate requested by the reserved resource traffic. Interleaving the two queues based on time did result in slightly better throughput than using a simple priority scheme. We expect the interleaving algorithm to do at least as well or better than the simple priority version, under almost all conditions. Improvements to the interleaving process are also possible. It may be better to compute a time into the future that should be used to determine the queue from which the next packet to be transmitted is chosen. This would reduce the probability of the "late" delivery of a VirtualClock packet that inadvertently gets behind a best-effort packet.

1 BACKGROUND

The advent of fast, low-cost computing power has radically changed the way networks are used. In the past, networks primarily supported non-real time data transfer in the form of electronic mail and file transfers, and some interactive applications such as text editing. These applications did not require that the network meet any hard throughput or delay constraints. Efforts were focused on techniques for getting the data to its destination with minimal loss and/or retransmission of packets. Congestion was a key concern, due to the low bandwidth of the connecting links. The current availability of audio and video applications, along with high-bandwidth networks, have forced a change in the requirements that networks need to satisfy. Networks must now guarantee that streams receive a certain level of performance. Current research, therefore, is focusing on solving this problem in the form of providing resource reservation for guaranteed and predicted service.

However, there is still a requirement to satisfy less demanding applications. We call traffic with such loose requirements *best-effort.* Historically, first-in, first-out (FIFO) queueing has provided this service within packet-switched networks. FIFO queueing, however, does not provide "fair" service. Users do not receive equal access to the available resources.* SFQ was developed to help solve this problem. SFQ, a probabilistic variant of fair queueing, uses a hash function to map packets into a fixed number of queues. It offers more equitable service to the available resources than FIFO queueing, without the processing and space overhead of strict fair queueing [Demers, Keshav, and Shenker 1989].

To satisfy all the demands on the network, i.e., guaranteed and best-effort service, a new hybrid algorithm must be created that combines the advantages of SFQ with a prototype resource reservation algorithm. This paper discusses the design and implementation of, and experimentation with, such an algorithm.

2 METHOD OF APPROACH

2.1 SYSTEM ARCHITECTURE

Figure 1 shows our basic architecture for combining SFQ with a resource reservation algorithm. A classifier examines each packet and determines if it requires best-effort or resource reservation service.The classifier then sends the packet to the appropriate service module. It is the service module's responsibility to determine how to best meet the service demands for this packet and place it in some kind of queueing mechanism that provides a dequeue routine to the next processing step. Historically, this next processing step is the driver to the network interface. In our design, a packet scheduler module intercedes to determine if a packet should be sent from the resource reservation module or the best-effort module. The packet scheduler module then passes the packet to the network interface driver.

2.2 CANDIDATE RESOURCE RESERVATION ALGORITHMS

Many different algorithms are currently being developed to satisfy the needs of traffic requiring reserved resources. These include

- JF Hierarchical Resource Management with Weighted Round-Robin Service [Jacobson and Floyd 1993; Floyd 1992]
- VirtualClock [Zhang 1989; 1990]

*Throughout this paper, fairness follows the max-min model developed by Bertsekas and Gallager. This model tries to maximize the allocation of each user, subject to the constraint that an increase in one user's (i's) allocation does not cause a decrease in some other user's allocation that is already as small as i's or smaller [Bertsekas and Gallager 1987].

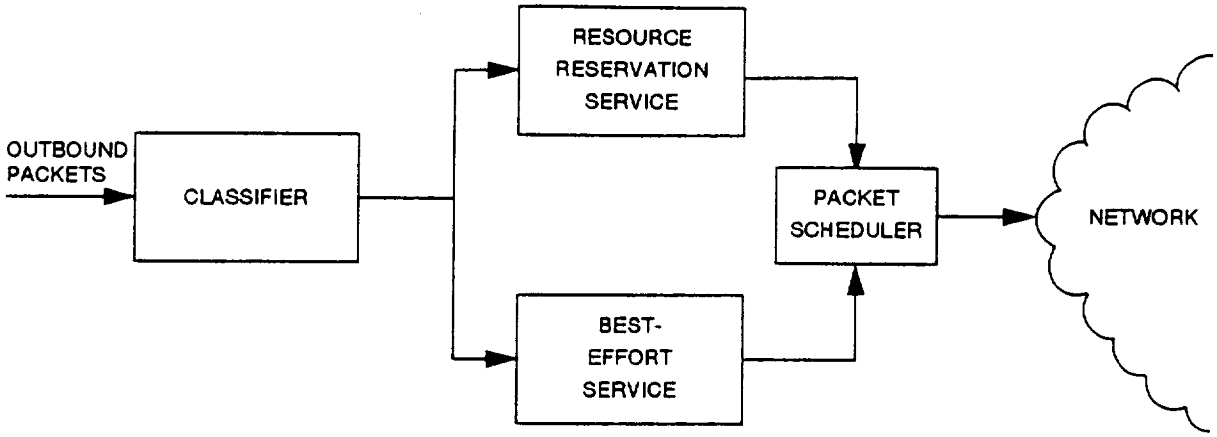


Figure 1. System Architecture

- Weighted Fair Queueing, as in the Integrated Services Packet Network (ISPN) [Clark, Shenker, and Zhang 1992].

However, many of the algorithms and supporting infrastructure are still under development and have not been released to the community or thoroughly documented. JF Hierarchial Resource Management is too immature: there is only a set of viewgraphs explaining some of the approach and no software has been released. VirtualClock algorithm has been released as a queueing mechanism used by ST-II [Topolcic 1990] in Bolt Beranek and Newman Inc.'s (BBN's) DARTnet kernel.* Weighted Fair Queueing, in the ISPN architecture, is also currently undergoing development and has not been released. Due to the time constraints of this project, therefore, VirtualClock has been chosen to support resource reservation. As a minimum, any resource reservation algorithm also requires a mechanism to supply the traffic characterization of the stream to the router, either explicitly or implicitly. ST-II will be used to supply this functionality.

2.2.1 VirtualClock

VirtualClock is a rate-based traffic control algorithm that maintains each user's average reserved rate by assigning a timestamp to every packet in each flow and ordering the output queue according to this timestamp. The algorithm is as follows:

```
1. For the first packet in flow$_i$, set its VirtualClock$_i$ to the real time
2. Upon receiving each packet from flow$_i$
   A. auxVC$_i$ = max(real time, auxVC$_i$)
   B. VirtualClock$_i$ = (VirtualClock$_i$ + Vtick$_i$) where Vtick$_i$ = 1/AR$_i$ (Average Rate
      in packets/second for flow$_i$) and auxVC$_i$ = (auxVC$_i$+Vtick$_i$)
   C. Stamp the packet with auxVC$_i$
3. Transmit packets from all flows by order of increasing timestamp
4. When there is no more buffer space, drop the last packet in the queue.
```

*All product names mentioned in this document are the trademarks of their respective holders.

To make sure the user's resource reservation is not exceeded, VirtualClock has an *AI* (Averaging Interval) for each flow, where the offered rate is compared to the reserved rate, *AR*. This rate is checked when the switch has received AIR_i packets, where AIR_i equals AR_i times AI_i. If the offered rate is greater than the reserved rate, (VirtualClock$_i$ - real time) > threshold, then some control actions may need to be taken. If the user is using less resources (VirtualClock$_i$ < real time), then VirtualClock$_i$ should be reset to the value of the real time clock. To handle priority flows, the algorithm above should be adjusted by (real time - *P*), where *P* represents the priority.

2.3 BEST-EFFORT TRAFFIC ALGORITHMS

FIFO queueing has been used to service best-effort traffic. It is efficient and simple to implement; however, it is "unfair." SFQ is superior to FIFO queueing in providing equal access to the available resources. It will therefore be used to service the best-effort traffic.

2.3.1 Stochastic Fairness Queueing

SFQ is a probabilistic variant of strict fair queueing. Instead of requiring that each flow have its own queue, SFQ has a fixed number of queues and uses a hashing function to map the IP source and destination address into one of the queues. Packets are entered into their assigned queues in a FIFO manner and are removed in a round-robin fashion between all nonempty queues (in the current implementation, the round-robin service is on a packet-by-packet basis). A seed to the hash function is occasionally perturbed, to allow a redistribution of the address pair mapping. This mapping redistribution ensures that flows are not consistently mapped into the same queue, so that a well-behaved source is not penalized by an ill-behaved source if the flows happen to map to the same queue at some point in time [McKenney 1991].

2.4 HYBRID ALGORITHM

To combine SFQ with VirtualClock requires defining the packet-scheduling algorithm that selects the queue from which the next packet will be transmitted. Two different approaches were explored. The first is a simple priority scheme where the VirtualClock queue always has priority over SFQ. The second approach is an attempt to provide "fairer" service. Reserved resource traffic should meet its scheduling requirements; however, this should not be done in a manner that could starve the best-effort traffic, as in the simple priority scheme above. (This starvation would occur in the case where the reserved resource traffic is high enough that there is always something in the VirtualClock queue.) The second approach, therefore, tries to interleave the packets in both queues while still guaranteeing the reserved resource traffic its average rate. To achieve this, we reinterpreted the timestamp calculated by VirtualClock for packet ordering as the time to send this packet as well. The scheduling algorithm, then, is as follows:

- If the VirtualClock queue is not empty, check the packet timestamp at the head of the VirtualClock queue.
    - If the packet timestamp is later than now and the SFQ queue is not empty, take the packet from SFQ; otherwise, take the packet from the VirtualClock Queue.
    - If the packet timestamp equals now or is earlier than now, remove the packet from the VirtualClock Queue.
- If the VirtualClock queue is empty, try to remove the packet from SFQ.

Note that this algorithm always tries to find a packet to send if there is one available. An improvement to the algorithm could be determining how far in the future packets should be sent from the VirtualClock queue before a best-effort packet is taken. This improvement would prevent any reserved resource traffic from being "late" because of a best-effort packet.

3 IMPLEMENTATION

The prototype implementation of this algorithm was done in version 13 of the DARTnet kernel,* using release 1.12 of BBN's implementation of ST-II and VirtualClock. The code supplied by BBN also included their traffic control interface abstraction, so we decided to try out their model in our implementation of the algorithm [Lynn 1993]. This interface consists of a series of routines that the algorithm designer must supply, based upon the requirements of their control algorithm. For our approach this consisted of defining initialization, classification, enforcement, enqueueing, and dequeueing. To enable the user to switch between different algorithms, a drain function is also supplied that empties both queues.

3.1 SOFTWARE DETAILS

The initialization procedure consists of calling vc_init_func and sfq_init_func, which allocate the proper data structures. From a functional description, the classifier for our approach is simple. If the traffic belongs to an allocated ST-II stream, then all packets belonging to that stream are sent to the VirtualClock queue. All other traffic is sent to SFQ. The classification function in the code, however, is currently NULL, due to the way the ST-II code is implemented. The ST-II packets are not mapped to a stream, i.e., classified until the enforce function, vc_enf_func, is called. At this point a RsrcClnt structure is allocated to the variable that contains the result of the classification function. Because our classification scheme is simple, assigning such a structure to best-effort traffic seemed overly cumbersome and a waste of space. The current implementation offered another mechanism for indicating what kind of service a packet required, so we decided to use it instead. There is now a new mbuf type called MT_TCDATA. The mbuf associated with a packet is assigned this type, if the packet needs to be stamped with a VirtualClock timestamp; otherwise, the mbuf is simply set to MT_DATA. Since this is exactly the information we needed, we decided to use the mbuf type for our classification mechanism. The enqueue routine calls the VirtualClock algorithm if the packet is type MT_TCDATA; otherwise, the SFQ enqueue routine is called. The dequeue routine uses the algorithm presented in Subsection 2.4.

3.2 TRAFFIC CONTROL INTERFACE ABSTRACTION

Our experiences in using the traffic control abstraction were positive; however, more work needs to be done to complete the abstraction. Our biggest problem in dealing with the current implementation was the lack of support for dealing with multiple queues. In standard BSD UNIX, the output queue associated with an interface is located in the ifq_head pointer of the ifnet structure. VirtualClock naturally uses this pointer as the place to store its queue. Unfortunately, the routines written to support SFQ also assume that this is the location for SFQ's main pointer. The current BBN interface extends the ifnet pointer to store more information, including some fields for storing

*A 4.1.1 Sun-OS based UNIX kernel.

traffic-control-specific data. We therefore used one of these pointers, s2p, to store the SFQ head pointer and modified the SFQ routines to use the pointer to this extended ifnet structure, aNetIF. However, this variable was the last available pointer. We therefore recommend that the number of these pointers should probably be extended to better serve more complex algorithms.

The abstraction also needs to be extended to include all queue manipulation routines. In particular, there is no provision for redefining the QFULL macro. This causes a problem for our algorithm, because one needs to look at two different data areas to determine if a queue is full; moreover, the algorithm for determining this information is different for each structure. Due to the way the driver was written, we were able to work around this problem by freeing the packet after a call to the enqueue routine of SFQ if the appropriate queue was full. However, this is not a good solution because other drivers may not be written the same way, and could result in system crashes if care is not taken in the installation. We also had problems with the driver because it would access the length associated with the queue in the ifnet structure and use that to determine if there was a packet available to transmit. This optimization created problems in our dual queue design because the counters for the SFQ structure were in a different area, so the presence of SFQ packets would not cause any packets to be sent to the interface. We could not easily merge the necessary counters into the ifnet structure area, because that area was being used by VirtualClock. Even though inefficiency may result, the information hiding between the driver and the queueing routines needs to be improved. We currently have removed this optimization of accessing the queue's data structure from the HSIS driver.

In implementing our algorithm, we also noticed the code was not completed for the Ethernet. There is no ability to call a different dequeue function. We believe this code was left incomplete because BBN did not have the source available and this kind of support was not necessary for VirtualClock. The incomplete code did not interfere with our test, because we designed our experiment to avoid needing to use the hybrid algorithm on the Ethernet interface.

The implementation of the classification mechanism for ST-II did not seem as modular as it could have been. As mentioned previously, classification does not happen until enforcement; so we needed to rethink the design for this function. We do not believe this should have been necessary.

Finally, to add SFQ to this abstraction only involved minor modifications to the existing SFQ code. Besides changing the location of the existing head pointer and allowing the routines to access the appropriate area in the ANetIF structure, we only needed to change the software to use routines instead of macros.

4 EXPERIMENTS

The experiments were executed on DARTnet, an ARPA research testbed network. DARTnet is a cross-country T1 network* that connects research sites via T1 tail circuits. The routers and most of the hosts are SPARC 1+, with the exception of two hosts, MM5 and Malarky, which are SPARC 2s. Figure 2 illustrates the portion of the network used in the experiment.

*Due to hardware constraints, the network operates at 1.334 Mb/s instead of 1.536 Mb/s.

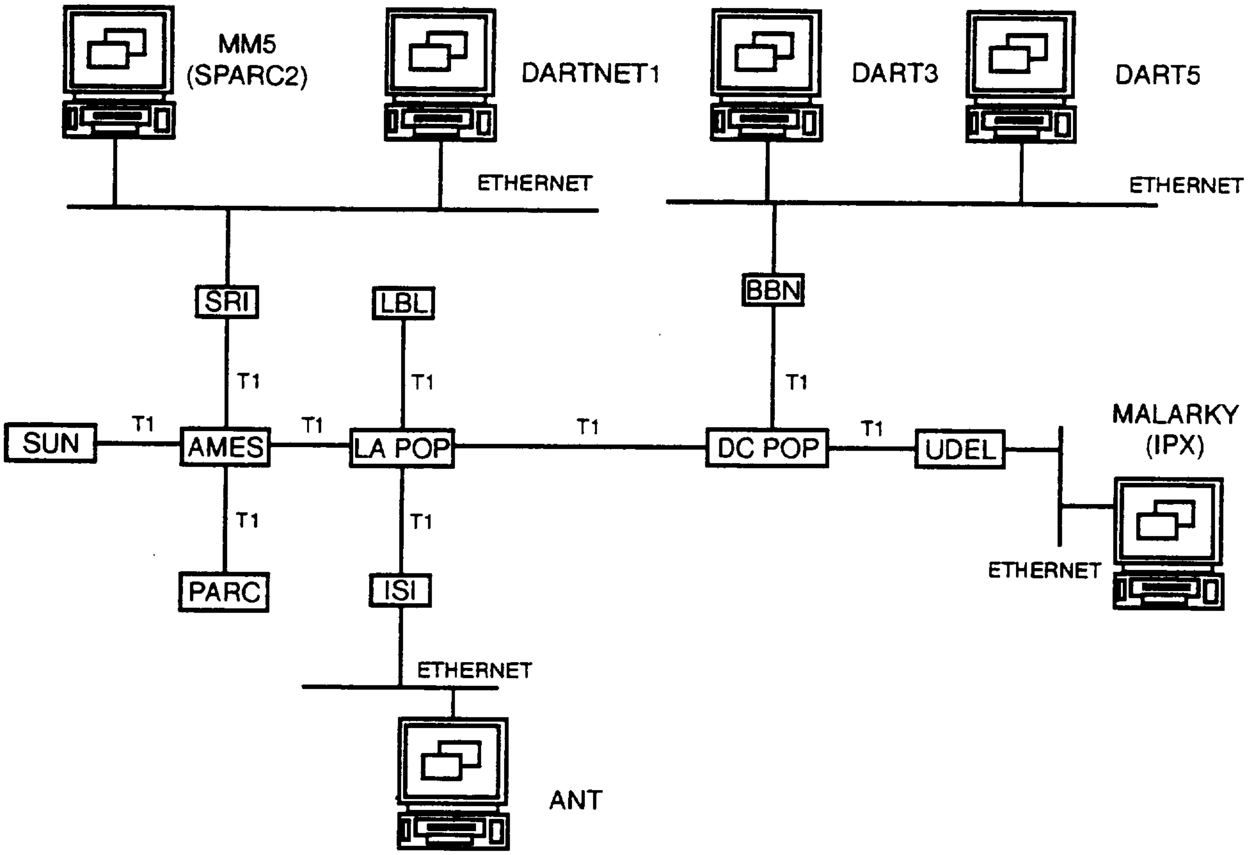


Figure 2. Experiment Topology

The experiment was run at least twice to verify its repeatability; however, in this report we will include only a single case. Each run of the experiment consisted of executing the experiment with three different traffic-control algorithms installed on the output interface of the T1 lines. Best-effort traffic, represented by UDP traffic, originated from machines whose queueing algorithm was FIFO. Reserved resource traffic, represented by ST-II streams, originated from machines that used VirtualClock as the queueing mechanism. Each stream in the experiment ran for 245 seconds, and the experiment was designed to overload the link so that the algorithm's behavior in times of congestion could be observed.

The DARTnet Traffic Generator (TG) was used to create the traffic streams. TG is an SRI-developed tool for creating high-quality and repeatable experiments on packet-switched networks. It executes as a source and sink program that enables experimenters to generate one-way traffic streams and gather statistical data about the transmission and reception of each stream. The TG is driven by a control language (script) that specifies different operating modes, protocols, addressing functions, traffic parameters, and execution times. At present, packet lengths and packet offer rates can be specified according to the following distributions: constant, uniform, exponential, and 2-state Markov. The measurements, therefore, are from user process to user process.

4.1 ALGORITHMS TESTED

To evaluate the performance of our hybrid algorithm, we choose to use the default algorithm in the traffic control interface that BBN supplied.* This baseline algorithm consisted of a single queue where VirtualClock and FIFO techniques were merged. Established streams in ST-II used VirtualClock while best-effort traffic used FIFO. The other two algorithms consisted of the versions of the hybrid algorithm described in Subsection 2.4. The next subsection provides more details on the implementation of these algorithms.

4.1.1 Configuration for the Baseline Algorithm

The VirtualClock algorithm, as implemented by BBN, varies from the description presented in Subsection 2.2.1. The main difference involves the actions taken to handle congestion. In the BBN implementation, packets are dropped as soon as the throughput allocated for the Averaging Interval is exceeded, rather than when the switch runs out of buffers. The throughput for an Averaging Interval is based upon the quality-of-service parameters specified by TG (average packet size and average packet rate). The current time period for the Averaging Interval is 1 second. Since BBN has an enforcement policy which prevents any stream for trying to send more than it is allowed during an interval, no control actions are necessary. Any best-effort traffic, i.e. traffic without an ST-II flow specification, is appended to the end of the VirtualClock packets in an FIFO manner. Thus, any best-effort traffic will resemble the results you would get with an FIFO queueing mechanism. There are no constraints to the depth of the queue when a VirtualClock packet needs to be enqueued. However, no best-effort traffic will be added after the queue contains more than 100 packets. There is also an admission control procedure for VirtualClock traffic; no stream will be accepted once the reserved capacity of the link has been totally allocated. This reserved capacity is currently set to 0.92 Mb/s on the 1.344 Mb/s line for DARTnet.

4.1.2 Configuration for the Hybrid Algorithm

As mentioned previously, the hybrid algorithm consists of two logical queues: one VirtualClock and one SFQ. Any traffic that would be handled by VirtualClock is treated in the same manner as the baseline algorithm. SFQ, however, has many parameters that can be tuned to improve its performance. These parameters include

- Individual queue depth
- Total number of queues
- Hash function.

The setting of these parameters for these experiments was somewhat arbitrary, because the experiment design was on a small enough scale that these factors did not affect the outcome significantly in most cases. More research needs to be done to determine the optimum choice for larger scenarios. The parameters chosen are described next.

*We originally planned to use a traffic control algorithm that only provided FIFO queueing and no resource management for our baseline measurements. However, we experienced problems setting up ST-II streams with TG for this algorithm. The streams could not be established, so we could not document this algorithm in this report. This appears to be an ST-II implementation problem in the kernel.

For the set of addresses used in the experiments, we chose an XOR type of hash function with seed perturbation enabled. The total number of queues was left to our default value of 257. Since this queue space is sufficiently large, we do not anticipate any collisions. We did verify the absence of collisions by checking for the first 200 changes of the seed for the addresses used: there were no collisions. The queue depth was set to 100 for each queue (i.e., each hash bucket). This parameter value was chosen to match the queue depth provided by the baseline algorithm used in this experiment.

Currently, SFQ supports only IP addresses. Due to the packet classification mechanism currently used, it may be possible that some ST-II traffic will use SFQ. Since the ST-II header and addressing format are completely different from those of IP, we chose to always assign any ST-II traffic arriving for SFQ to queue 256, where raw traffic also is queued. This assignment is not expected to interfere with our results, since the amount of ST-II traffic arriving for SFQ is expected to be small.

4.2 EXPERIMENT DESCRIPTION

4.2.1 Objective and Procedure

This experiment was designed to demonstrate the two key design goals of our hybrid algorithm. In particular, these goals are that

- Reserved resource traffic streams acquire their requested average packet generation rate
- Best-effort traffic streams receive the benefits of SFQ (streams receive equal portions of the available bandwidth, and hosts sending under their allocation are rewarded).

In this experiment, four different streams were used; two streams of UDP, i.e. best-effort, traffic and two streams of ST-II, i.e., reserved resource, traffic. The reserved resource streams were slightly overbooked in their reservation, to ensure sufficient resources at all times. The best-effort traffic was used to overload the network. Table 1 and Figure 3 provide a description of the traffic conditions during the time of the experiment.

4.2.2 Data Description

For the reserved resource traffic, the graphs in the Appendix show the offered and received throughput, and the offered and received generation rate. For the best-effort traffic, the graphs show the offered and received throughput. The naming conventions used in the subtitles indicate the algorithm used: VC stands for the baseline algorithm; VC_P_SFQ indicates VirtualClock with priority over SFQ; and VC_SFQ indicates interleaving based on time to send.

Table 1. Traffic Conditions

SOURCE	DESTINATION	TYPE OF TRAFFIC	ARRIVAL DISTRIBUTION	PACKETS PER SECOND*	PACKET SIZE IN BYTES (INCLUDING HEADERS ABOVE NETWORK LAYER)	PERCENTAGE OF BANDWIDTH†
DARTNET1	DART3	Reserved	Constant	22	782	10
SUN	DART5	Reserved	Constant	66	782	30
MM5	MALARKY	Best-effort	Exponential	130	782	60
PARC	ANT	Best-effort	Exponential	43	782	20

*The numbers of packets are approximate.
†Percentages are approximate.

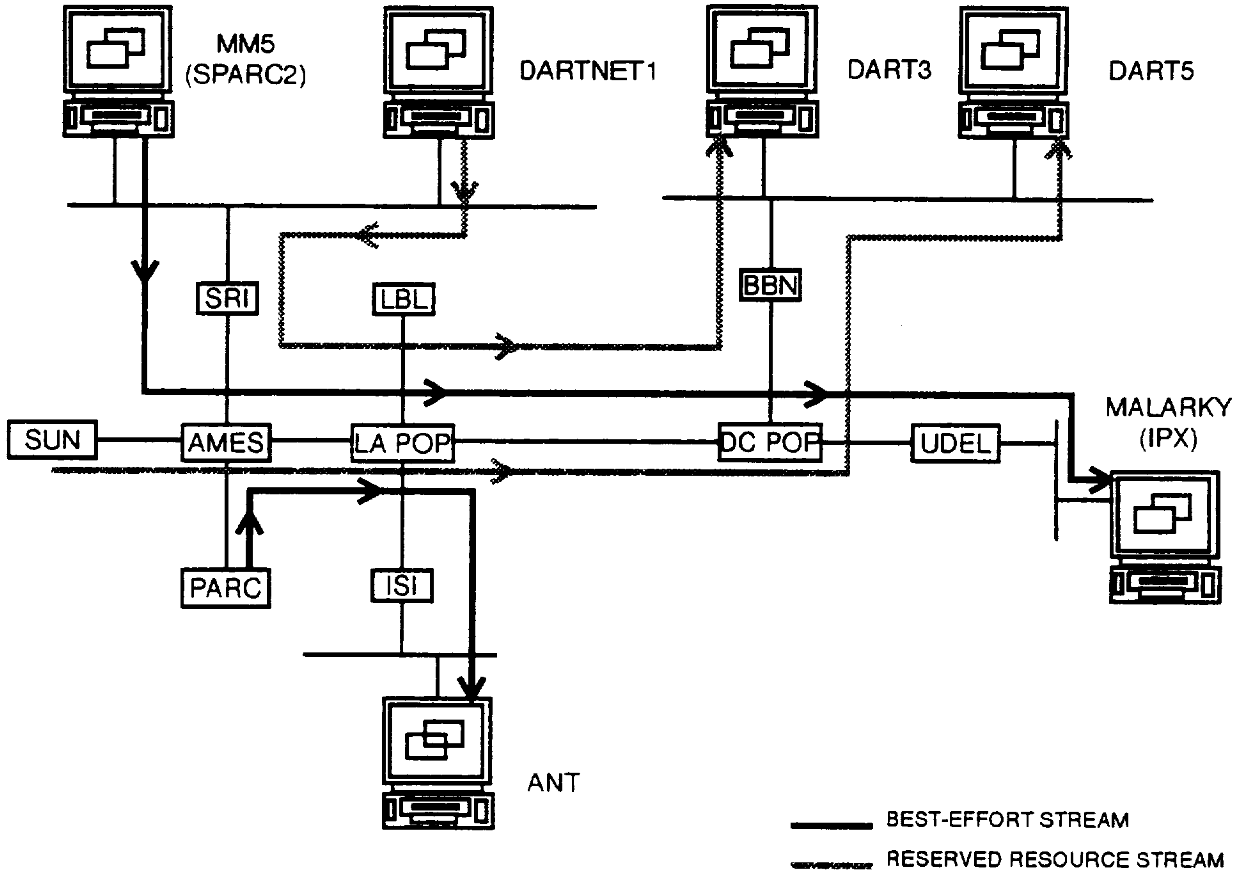


Figure 3. Experiment Traffic Flow

Table 2 summarizes the results of the experiment with the reserved resource traffic, including the average offer rate as a packet generation rate, average receive rate, average offer rate as a percentage of bandwidth, average throughput, average delay, and delay variance for each stream.

Table 2. Traffic with Reserved Resources

STREAM	ALGORITHM USED	AVERAGE PACKET GENERATION RATE (packets/s)	AVERAGE PACKET RECEIVE RATE (packets/s)	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (s)	DELAY VARIANCE (s)
DARTNET1 to DART3	VC	21.9417	19.4153	10.21	9.04	0.0873	0.0004
	VC _P_SFQ	21.9408	21.9387	10.21	10.21	0.0745	0.0000
	VC _SFQ	21.9399	21.9367	10.21	10.21	0.0796	0.0000
SUN to DART5	VC	65.8046	52.4324	30.63	24.40	0.0742	0.0002
	VC _P_SFQ	65.8075	65.8039	30.63	30.63	0.0717	0.0000
	VC _SFQ	65.8071	65.8023	30.63	30.63	0.0709	0.0000

Table 3 summarizes the results of the experiment with the best-effort traffic, including the average offer rate as a percentage of bandwidth, average throughput, average delay, and delay variance for each stream.

Table 3. Best-Effort Traffic

STREAM	ALGORITHM USED	AVERAGE OFFER RATE (percentage of 1.344 Mb/s)	AVERAGE THROUGHPUT (percentage of 1.344 Mb/s)	AVERAGE DELAY (s)	DELAY VARIANCE (s)
MM5 to MALARKY	VC	60.26	48.02	6.2137	0.9580
	VC _P_SFQ	60.26	34.04	1.4468	0.0219
	VC_SFQ	60.26	35.51	1.3608	0.0200
PARC to ANT	VC	20.02	15.29	6.1479	0.9836
	VC _P_SFQ	20.01	20.01	0.0762	0.0010
	VC_SFQ	20.02	20.01	0.0708	0.0008

4.2.3 Results

The results of this experiment show that both versions of the hybrid algorithm satisfy the reservation requests and deliver fairer service for best-effort traffic than the baseline algorithm. (The best-effort stream that demanded 20% of the remaining 60% received its fair share of 20% under our hybrid algorithm. Under the baseline algorithm, it received only ~15%, or about 76% of its requested throughput; therefore, this stream did not receive a fair share of the available bandwidth). The best-effort traffic also received better service under the hybrid algorithm than the baseline algorithm; the average delay and variance in delay was significantly lower.

The two versions of the hybrid algorithm behaved almost identically in our experiment. Interleaving the two queues based on time did result in slightly better throughput for the one best-effort stream, MM5 to Malarky, that could vary in the amount of bandwidth it received. We expect the interleaving algorithm to at least do as well or better than the simple priority version under almost all conditions.

The failure of the baseline algorithm to deliver the requested rate of service for the reserved resource traffic is probably not significant. We believe there is a bug in the ST-II implementation, because the bottleneck node, AMES, reported memory allocation problems during all attempted executions of the baseline algorithm. Resolving this problem, however, was beyond the scope of our project.

5 CONCLUSION

Combining SFQ with VirtualClock can effectively satisfy the demands of resource reservation traffic while providing best-effort traffic with equal access to the remaining bandwidth. Interleaving the packets from the VirtualClock queue and SFQ seems to be better than a simple priority scheme. Interleaving provides fairer access to the network by allowing best-effort traffic to be sent as soon as possible, without interfering with the scheduling requirements of the reserved resource traffic, and also achieved slightly better throughput than a simple priority scheme. The interleaving algorithm could probably be improved by calculating the window during which the next virtual packet should be sent, so as to prevent "late" delivery of the packet. The baseline algorithm, which gave priority to VirtualClock packets over FIFO best-effort packets, can result in poor performance for the best-effort traffic. Finally, the generic traffic control abstraction that BBN is developing provided a good framework for developing the algorithm. However, working with multiple queues was not as easy as anticipated. The abstraction of the queueing mechanism was not complete, and lack of information hiding in the driver caused some initial bugs.

REFERENCES

Bertsekas, D., and R. Gallager. 1987. *Data Networks*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Clark, D., S. Shenker, and L. Zhang. 1992. "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," *Proc. of ACM SIGCOM*, pp. 14-26.

Demers, A., S. Keshav, and S. Shenker. 1989. "Analysis and Simulation of a Fair Queueing Algorithm," *Proc. of ACM SIGCOMM*, pp. 1-12.

Floyd, S. 1992. "A Report on Link-Sharing Experiments," draft paper, Lawrence Berkeley Laboratory, Berkeley, California.

Jacobson, V., and S. Floyd. 1993. *Hierarchical Resource Management*, unpublished work, Lawrence Berkeley Laboratory, Berkeley, California.

Lynn, C. 1993. "Net Interface Extension Memo," draft technical note, Bolt Beranek and Newman Inc., Cambridge, Massachusetts.

McKenney, P. 1991. "Stochastic Fairness Queueing," in *Internetworking: Research and Experience*, Vol. 2, pp. 113-131.

Topolcic, C., ed. 1990. "Experimental Internet Stream Protocol, Version 2 (ST-II)," RFC 1190.

Zhang, L. 1989. "A New Architecture for Packet Switching Network Protocols," Doctoral dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Zhang, L. 1990. "VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks," *Proc. of ACM SIGCOMM*, pp.19-29.

Appendix F

AN ANNOTATED BIBLIOGRAPHY FOR CONGESTION CONTROL AND RESOURCE RESERVATION

Technical Report •March 1993

ANNOTATED BIBLIOGRAPHY FOR CONGESTION CONTROL AND RESOURCE RESERVATION

Diane S. Lee, Sr. Research Engineer
Barbara A. Denny, Computer Scientist
Information and Telecommunications Sciences Center

Project 8600
ITAD-8600-TR-93-74

Prepared for:

NASA Ames Research Center
Moffett Field, California 94035

Attn: Dr. Henry Lum, Code RI, M/S: 244-7

and

Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, Virginia 22203-1714

Attn: Dr. Paul Mockapetris

Approved by:

Boyd C. Fair, Director
Information and Telecommunications Sciences Center

Michael S. Frankel, Vice President and Director
Information, Telecommunications, and Automation Division

CONTENTS

1 INTRODUCTION

SRI International (SRI) has prepared an annotated bibliography of recent articles involving congestion control in high-speed networks, and resource reservation protocols by DARTnet[*] researchers. A synopsis of each article is provided, along with an evaluation of the presentation and a categorization of the information presented. After surveying the collection, we determined that each article belongs to one of four categories: tutorial, overview, model/methodology, or technical approach/solution to the problem space. The assignment of each article to a category is summarized below.

- Tutorial
  - Jain, 1990
- Overview
  - Jain, 1991
  - Kurose, 1993
  - Maxemchuk and Azrki, 1990
- Model/Methodology
  - Mukherjee and Strikwerda, 1991
  - Parulkar and Turner, 1990
- Technical Approach/Solution
  - Bala, Cidon, and Sohraby, 1990
  - Clark, Shenker, and Zhang, 1992
  - Floyd and Jacobson, 1992
  - Golestani, 1990
  - Lai, 1990
  - Mitra, 1990
  - Ramkrishnan and Jain, 1988
  - Shacham and McKenney, 1990
  - Williamson and Cheriton, 1991
  - Zhang, 1990
  - Zhang, Deering, Extrin, Shenker, and Zappala, 1993.

[*] DARTnet is a T1 testbed network sponsored by DARPA.

2 BIBLIOGRAPHY

2.1 JAIN, 1990

This paper summarizes Raj Jain's article entitled "Congestion Control in Computer Networks: Issues and Trends," which appeared in the May 1990 issue of *IEEE Network Magazine*. The article can be found on pages 24 through 30.

2.1.1 Summary

This paper discusses the following topics:

- Myths about congestion control
- Classification of congestion problems and solutions
- Requirements and policies affecting the design of congestion control schemes
- Proposals for congestion control schemes
- Topics for research.

Jain first clears up some myths about congestion control. He states that congestion is not caused by shortages of buffer space, slow links, or slow processors; increases/improvements of these resources will not cause congestion to go away. Congestion is not a static resource shortage problem; it is a dynamic resource allocation problem.

Jain goes on to classify congestion problems and solutions by first defining congestion (i.e. during any time interval, congestion occurs when the total sum of demands on a resource is more than its available capacity). Congestion schemes are then categorized into resource creation and demand reduction schemes. The latter scheme includes service denial, service degradation, and scheduling. Both resource creation and demand reduction schemes require a feedback and control element. A description of several feedback mechanisms is provided (e.g., feedback messages, feedback in routing messages, rejection of further traffic, probe packets, and feedback fields in packets) and alternatives for the location of control are discussed (e.g., transport layer, network access, network layer, and data link layer). Jain emphasizes that the control and feedback rates for congestion control schemes should be similar; otherwise the system will have oscillatory or unresponsive behavior.

The requirements that congestion control schemes must meet are briefly reviewed. The schemes must have low overhead, be fair, be responsive, work in unfavorable environments, and be socially optimal (e.g., *total* network performance must be optimized). The author goes on to discuss architectural/implementation issues that affect congestion control schemes. These include network policy (e.g., connection oriented versus connectionless); packet queuing; packet drop; route selection; packet lifetime; round-trip delay estimation and timeout-interval algorithms; packet retransmission; packet acknowledgment; flow control (window based versus rate based); and buffer management.

Next, the author describes three congestion schemes that the Digital Equipment Corporation (DEC) has recently proposed. References are also provided that detail the following schemes:

- Time-Out Based Congestion Control is based on the idea that packet loss is a good indicator of congestion; thus, on a packet timeout, network load should be reduced. Three proposals are given that differ by the packet window sizing algorithm (parabolic increase, slow start, and linear increase.)
- DECbit* Scheme for Congestion Avoidance uses a single bit in the network header. (The discussion distinguishes between congestion avoidance and congestion control schemes).*
- Delay-Based Scheme for Congestion Avoidance measures delay and adjusts the traffic depending upon the delay.

These three schemes do not require any additional packets, and all parameters are dimensionless (i.e., the schemes do not use any parameters).

Finally, Jain provides his assessment of areas of research in congestion control including path splitting, heterogeneous networks, dynamic link creation, server congestion, and integrated networks with voice and data.

2.1.2 General Comments

This paper is a good tutorial on congestion control, providing definitions and discussing problems, general solutions, and future areas of research. This is not a technical article–in most cases Jain refers to papers that detail the topics he briefly discusses. There is an extensive list of references, many of which are articles from recent journals.

2.1.3 Categorization

General tutorial on congestion control.

*All product names mentioned in this document are the trademarks of their respective holders.

2.2 JAIN, 1991

This paper summarizes Raj Jain's technical report entitled "Myths About Congestion Management in High-Speed Networks." It was published by Digital Equipment Corporation in January 1991 under the number DEC-TR-726.

2.2.1 Summary

In addition to discussing myths about congestion control and traffic patterns on high-speed networks, Jain's paper looks at the pros and cons of several congestion control and avoidance techniques. Techniques discussed include:

- Window versus rate control
- Router-based versus source-based controls
- Back pressure
- Prior reservation versus walk-in.

Jain concludes with a discussion of a congestion management strategy.

Jain first clears up some myths about congestion control. He states that congestion is not caused by shortages of buffer space, slow links, or slow processors; increases/improvements of these resources will not cause congestion to go away. Congestion is not a static resource shortage problem; it is a dynamic resource allocation problem. Jain also describes myths regarding the appropriateness for high-speed networks of the congestion control and avoidance techniques listed above. Jain argues that the key issue in the design or selection of a congestion management scheme is the traffic pattern that depends upon the application; the design should accommodate the characteristics of a number of applications, simultaneously.

The following briefly highlights the issues Jain presents for each of the congestion management techniques listed above. Jain's discussions are backed up by references to work performed by other researchers.

2.2.1.1 Window Versus Rate Control

Most current network architectures utilize window mechanisms for flow control; however, the author advocate the use of rate-based controls for high-speed networks. Window-based flow control schemes originated from the desire to keep memory from overflowing; but as memory is no longer the bottleneck, other schemes should be considered. Problems with window-based schemes are noted. With some implementations of window-based controls, all packets of a window can be transmitted back to back resulting in bursty traffic. Additionally, much high-speed traffic is stream oriented, requiring a guarantee based on rate rather than count.

There are, however, misconceptions about rate-based controls. It is not generally understood that specifying the rate requires two quantities (the number of packets, n, over a period T), nor is it understood that rate-based controls are hop-by-hop mechanisms that cannot be enforced end-to-end unless all intermediate systems are aware of the rate parameters and are required to enforce them. Finally, with dynamic rate-based controls there is a possibility of significant increase in packet queue lengths when the total input rate is close to the capacity.

2.2.1.2 Router-Based Versus Source-Based Controls

With source-based control schemes, information is fed back to the sources (end systems) that initiate remedial control action: examples include slow-start, DECbit, and Congestion Using Timeout at the End-to-end layer (CUTE). Arguments presented against source-based routing include (1) significant delay between sensing the congestion and taking remedial action, (2) lack of cooperation by sources, and (3) injection of additional packets into the network.

Although router-based control schemes do not suffer from problems characteristic of source-based schemes, they do have their problems. The key problem is that they introduce complexity in the intermediate nodes (routers) that are shared resources. Jain discusses schemes that reduce this complexity (e.g., implicit feedback schemes, and specially coded packets). Examples of router-based controls are random-drop policy, fair queueing, and backpressure.

Jain concludes that router-based controls are required for fairness and work under short-duration overloads, while source-based controls are required for longer overloads.

2.2.1.3 Back Pressure

Back pressure is a form of hop-by-hop, on/off flow control. Jain argues that back pressure is a data link-level mechanism, and should be used only for short-duration overloads. For long-duration overloads, back pressure should be supplemented with a transport-level or network access level control scheme.

2.2.1.4 Prior Reservation Versus Walk-In

Prior reservation has advantages and disadvantages. It is preferred if bandwidth or delay guarantees are difficult to achieve with walk-in service. Additionally, prior reservation makes resource management easier, since demands and capacities are known in advance. On the other hand, prior-reservation schemes waste unused reserved resources and incur setup overhead. Jain concludes that reservation is good for long, steady sessions, while walk-in service is required for short bursty traffic and is suitable for highly dynamic environments.

Finally, Jain argues that the type of congestion control scheme needed is dependent upon the duration of the overload; and every network can have overloads of all durations, every network needs a combination of controls at various levels. As a general rule of thumb, the longer the duration, the higher the layer at which control should be exercised.

2.2.2 General Comments

This paper provides good comparisons and arguments for and against a number of techniques for handling congestion in high-speed networks. It is tutorial in nature, in that it is not very technical and attempts to cover many topics in congestion management. In many cases, references are made to papers that detail the topics briefly discussed. There is an extensive list of references, many of which are articles from recent journals.

2.2.3 Categorization

General tutorial playing devil's advocate for many techniques for handling congestion control.

2.3 KUROSE, 1993

This paper summarizes Jim Kurose's article entitled "Open Issues and Challenges in Providing Quality of Service Guarantees in High-Speed Networks," which appeared in the January 1993 issue of *Computer Communication Review*, Volume 23, Number 1. The article can be found on pages 6 through 15.

2.3.1 Summary

Kurose identifies the challenges and open issues involved in providing quality-of-service (QOS) guarantees to sessions in a high-speed wide area network and briefly surveys research in this area. After providing definitions and discussing fundamental challenges associated with QOS guarantees, the author describes and discusses four approaches towards providing QOS guarantees: (1) tightly controlled, (2) approximate, (3) bounding, and (4) observation based.

Kurose begins by describing the problem. Unlike traditional data networks, future high-speed networks will be required to carry a broad range of traffic classes. These networks will have to do so while providing a guaranteed performance or quality of service to many of these traffic classes. The actual QOS performance metrics vary from one application to another, but are likely to include such measures as cell loss, delay, and jitter guarantees. The need to provide end-to-end QOS guarantees while taking advantage of the resource gains offered by a statistically multiplexed transport mechanism remains an important, largely unsolved problem. The complexity of this issue can be seen by the simple question that must be answered each time a call/session arrives: "*Can the requested call be accepted by the network at its requested QOS, without violating existing QOS guarantees made to ongoing calls?*"

The author distinguishes QOS guarantees by *deterministic* guarantees and *statistical* guarantees. In the deterministic case, guarantees provide a bound on the performance of all cells within a session. Statistical guarantees on the other hand, promise that no more than a specified fraction of cells will see performance below a certain specified value. A number of references are provided that discuss and compare these QOS categories.

The challenges posed by characterizing performance (and thus providing QOS guarantees) are briefly explained by the author. First, sources of traffic such as packetized voice and video exhibit correlated, time-varying behavior that is significantly more complex than that of traditional data network sources. Second, since QOS requirements are defined on an individual, per-session basis, it is no longer sufficient to simply determine the performance of the aggregated network traffic. And finally, the complex interactions among sessions must be considered if they interfere with each other while they pass through various network nodes. An illustration is provided of the later issue, concluding that to provide guarantees, an algorithm must either (1) provide multiplexing mechanisms that avoid changes in traffic characteristics when sessions are multiplexed together (i.e., approach 1, above), or (2) characterize the increases in peak rates, as well as the other changes in a session's traffic characteristics, that occur as a result of multiplexing (i.e., approaches 2, 3, and 4, above.)

The latter half of the paper discusses the pros and cons, and issues relating to these approaches, and are summarized below.

Tightly Controlled approaches use a non-work-conserving multiplexing (queueing) discipline to ensure that an individual session's output traffic characteristics are the same as that session's input traffic characteristics.

- Pro: The characteristics of traffic are preserved as it passes through the network, and consequently performance bounds can be computed in a simple manner.
- Con: A fairly sophisticated, non-work-conserving queueing discipline must be implemented that tracks each individual session's timing requirements on a per-session basis.
- Con: A session admitted to the network essentially "reserves" bandwidth based on its peak rate, potentially leaving the links significantly under utilized.

Approximate approaches are characterized by relatively "simple" models for traffic sources at the network's edge (and within the network).

- Pro: Simplicity makes these approaches well suited for real-time, on-line implementations.
- Pro: This approach takes advantage of statistical multiplexing gains, potentially carrying traffic whose peak rate exceeds link capacity.
- Con: QOS computations are "approximate," tending to be conservative.
- Issue: It is unknown to what extent more complicated sources (voice and video, for example) can be characterized by the relatively simple approximate models such as an on/off source.
- Issue: A potential concern is that traffic models, whether at the source or deep within the network, require some form of Markovian assumptions.

Bounding approaches explicitly account for the fact that a session's traffic does indeed change each time it passes through a work-conserving multiplexer.

- Issue: For statistical bounding approaches, a concern is the extent to which traffic can be characterized by the form of the distributional bounds required by various approaches.
- Issue: For statistical and deterministic approaches, a concern is their reliance on the ability to bound the maximum length of each queue's busy period for a given set of traffic specifications.

Observation-Based approaches use previously made measurements of certain types of traffic sources to characterize an arriving call and in determining the call acceptance decision.

- Pro: Using a predictive service may result in a network that is more fully utilized.
- Issues: There are many research topics, including the effects of different measurement/estimation techniques on the protocol, the overhead involved in measurement, the influence of the number of multiplexed sessions on the reliability of the guarantees, and a thorough study of the mechanism in a large network environment.

2.3.2 General Comments

This paper provides a broad description of four approaches for providing quality of service guarantees. This is not a technical article; in most cases Kurose refers to papers that detail the topics he briefly discusses. There is an extensive list of references, many of which are articles from recent journals.

2.3.3 Categorization

Discussion of issues in providing quality-of-service guarantees and a brief survey of research in this area, focussing on approaches.

2.4 MAXEMCHUK AND EL AZRKI, 1990

This paper summarizes Nicholas Maxemchuk's and Magda El Azrki's article on "Routing and Flow Control in High Speed Wide Area Networks," which appeared in the *Proceedings of the IEEE* in January 1990. The article can found in Vol. 78, No. 1, on pages 204 through 221.

2.4.1 Summary

This paper emphasizes routing and flow control in high-speed, wide-area, store-and-forward networks and begins with a discussion of the following:

- Survey of routing and flow control for wide-area, local-area, and metropolitan networks
- Classification of routing and flow control mechanisms
- Predictions for high speed networks

The authors begin by briefly comparing the characteristics of and differences between local, metropolitan, and wide-area networks, including the cost of transmission facilities and topologies.

A history of routing and flow control mechanisms is presented for local, metropolitan, and wide-area networks. Features, deficiencies, and adaptations of various mechanisms are presented for a number of topologies. Mechanisms discussed include

- Routing: random, flooding, hot potato, shortest queue plus bias, delta, proportional, dispersity, deflection
- Flow Control: windowing (end to end, hop by hop), is arithmic, token passing, random access, slotted systems.

Routing and flow control mechanisms are usually categorized by characteristics such as centralized versus distributed control, stochastic vs. deterministic decisions, global vs. local information, and topological dependence. For analyzing high speed networks, the authors have classified routing and flow-control mechanisms according to the amount of processing required and bandwidth utilization. Bandwidth utilization includes exchange of control information or additional data bits transmitted. The amount of processing required reflects processing performed in the intermediate nodes (i.e., at the rate of the network) and processing performed as a packet enters the network, or leaves the network (i.e., at the rate of the source.) This classification is further developed to portray the cost of routing and flow control mechanisms in terms of processing at the rate of the network and bandwidth inefficiency.

As a result of this classification, the authors make recommendations for high-speed, wide-area networks. These include

- Routing
  - If an adaptive mechanism is used, it should be a scheme that adapts on a session basis.
  - Datagram routing techniques that adapt using local information, including hot-potato routing and its derivatives (e.g., shortest queue plus bias, delta, and deflection routing), do, however, appear to have characteristics that are well suited for high-speed networks. In this class of mechanisms, deflection routing appears to be the best choice.

  - Flooding is expensive in terms of both processing and bandwidth, and should not be considered.
  - Fixed routing schemes that distribute data over a larger part of the network, including dispersity and proportionally routing, look attractive.
- Flow Control
  - Of the hop-by-hop window mechanisms, Beeforth's virtual circuit scheme appears to be best suited to high-speed networks.
  - For end-to-end mechanisms, selective repeats appear to be best suited for high-speed networks.
  - Single token passing and random access mechanisms do not scale well to high-speed networks.
  - The slotted access mechanism appears to be well suited to high-speed networks, requiring very little processing at the network rate and making reasonably efficient use of bandwidth. Slotted schemes do not ensure fair access; if used they should be combined with another scheme to negotiate rates for particular sources.

2.4.2 General Comments

This is an excellent article. A survey of routing algorithms and flow control mechanisms for local-area, metropolitan-area, and wide-area networks is given along with a classification of these algorithms' mechanisms. The descriptions of the algorithms and mechanisms are not very technical, but the essence of each is clearly presented. Comparison of the algorithms and mechanism is well done via graphs and is then used to justify recommendations for high-speed, wide-area networks. References are given for all the routing and flow control mechanisms discussed.

2.4.3 Categorization

This article surveys routing algorithms and flow control mechanisms for local-area, metropolitan-area, and wide-area networks, with emphasis on high-speed, wide-area, store-and-forward networks.

2.5 MUKHERJEE AND STRIKWERDA, 1991

This paper summarizes the technical report "Analysis of Dynamic Congestion Control Protocols–A Fokker-Planck indent Approximation," by A. Mukherjee and J. Strikwerda. It was published by the University of Wisconsin-Madison Computer Sciences Department in February 1991. The report number is 1003.

2.5.1 Summary

This technical report presents an analysis model and methodology, based on a queuing system, to evaluate the performance of a wide range of feedback control schemes. The model is an extension of the classical Fokker-Planck equation and is motivated by proposals for adaptive, window-based congestion control algorithms by Jacobson-Ramakrishnan-Jain (JRJ). The authors investigate the performance of congestion control protocols that dynamically change input rates based on feedback (implicit or explicit) information received from the network. Issues of stability, convergences (or oscillations), fairness, and the effect of delayed feedback on performance are addressed by the model.

Specifically, the paper presents the model, motivates the analysis methodology, and derives a Fokker-Planck approximation for the time dependent queue behavior; discusses the properties of the JRJ algorithm when only one source is using the resource; investigates the properties of the system with multiple sources; investigates these properties in the presence of delayed feedback; and finally presents some conclusions. Given the high mathematical content of this paper, only a brief description of the model is provided here, along with a simple description of the results of the authors' theoretical analysis. Emphasis is given to the authors' conclusions.

2.5.1.1 Model

The model the authors have chosen is motivated by the JRJ algorithm for window adjustment. In the JRJ algorithm, when congestion is detected, the window size is decreased multiplicatively, and is increased linearly when there is no congestion. Although this algorithm is intuitive, the authors argue that it is not clear what values the parameters of the JRJ algorithm should take, nor is it provably clear if the algorithm is fair or stable. To understand the behavior of dynamic congestion control algorithms, the authors study a queuing system with a time varying input rate. The input rate is adjusted periodically on the basis of some feedback that the end-point receives about the state of the queue. Time evolution of the queue length density function is analyzed.

The authors present the following model, which is used to describe the behavior of generic rate-control algorithms in their analysis:

$$\frac{d\lambda}{dt} = \begin{cases} +C_o & if\ Q(t) \le \bar{q} \\ -C_1\lambda & if\ Q(t) > \bar{q} \end{cases}$$

where $\lambda(t)$ is the arrival rate based on the current queue length $Q(t)$ at some bottleneck node, $\bar{q}$ is some target queue length, and C_0 and C_1 are positive constants. This models a linear increase and exponential decrease in λ, similar to JRJ.

2.5.1.2 Theoretical Results

The authors analyze their model by evaluating the time-dependent behavior of the joint probability density function of the queue length and the instantaneous queue growth rate. The following properties of this model are proved.

- The model converges in the limit with the limit points q equal to the target length queue and λ equal to the average service rate of the queue.
- When extended to accommodate multiple sources, the model is fair in that all sources sharing a resource get an equal share of the resource.
- With feedback delay, the model introduces oscillations. These oscillations converge to a limit cycle (i.e., a cyclic pattern that is constant in the limit.)

2.5.1.3 Behavior of Some Existing Congestion Control Protocols

Given the analysis of the queuing model, the authors conclude the following regarding the Jacobson-Ramakrishnan-Jain adaptive window algorithm. In the absence of feedback delay, senders using the JRJ algorithm converge to an equilibrium. The algorithm is fair in that all sources sharing a resource get an equal share of the resource if they use the same parameters for adjusting their rates. A delay in the feedback information introduces cyclic behavior. If different sources get the feedback information after different amounts of delay, the algorithm may also be unfair (i.e., the sources may get unequal throughput). These results strengthen the observations in previous studies (Zhang, and Bolot and Shankar) and also identify the underlying reasons. For instance, if the adaptive algorithm is linear-increase/exponential-decrease, then oscillations are due to delayed feedback. However, if the adaptive algorithm is linear-increase/linear-decrease, then the oscillations are due to both the algorithm itself and the delay in the feedback path.

The authors have demonstrated a methodology that can lead to better understanding of a wide range of feedback control schemes.

2.5.2 General Comments

This paper is different from the others in this bibliography in that it does not propose a specific control algorithm, but presents a model algorithm that is then used to help provide a solid mathematical understanding of the behaviors of existing congestion-avoidance algorithms such as the work by Jacobson, Ramakrishnan, and Jain. The paper is very mathematical.

2.5.3 Categorization

Analysis methodology for evaluating the behavior of existing congestion-avoidance protocols (covering most window-based schemes).

2.6 PARULKAR AND TURNER, 1990

This paper summarizes the article "Towards a Framework for High-Speed Communication in a Heterogeneous Networking Environment," by Gurudatta M. Parulkar and Jonathan S. Turner. It appeared in the March 1990 issue of *IEEE Network Magazine* on pages 19 through 27.

2.6.1 Summary

This paper describes a framework for high-speed communication for a complex heterogeneous networking environment consisting of autonomous and/or technologically dissimilar subnetworks. The authors provide some background material on the ARPA Internet model and on high-speed packet switching. This background motivates their discussion of an extended internet model that includes a connection-oriented internet protocol, an internet addressing scheme, a parametric description of subnet capabilities, and design and implementation issues. A brief discussion of end-to-end and host interface issues is also presented.

As background material, both strengths and weaknesses of the current Internet are described. Weaknesses pointed out include insufficient raw bandwidth to support increased traffic and new applications such as graphics and multimedia; the inability of the packet switching technology to effectively use high bandwidths (100 Mb/s and more); the Internet's inability to support a predictable level of performance for its applications; and lack of central management.

To provide flexible use of the bandwidth required to support applications ranging from low-speed data to voice, video, and high-speed data, research in high-speed packet switching has focussed on high-performance digital transmission facilities, hardware implementations, and connection-oriented models. The authors briefly describe their connection-oriented switching model, which focuses on problems associated with multipoint communication where a single connection may include a large number of end points. Their connection abstraction allows a multipoint connection to have more than one channel and each channel to have different bandwidth parameters and access permissions.

The authors describe the following extensions to the Internet model to allow operation in a heterogeneous environment:

- A connection-oriented transport service at the internet level to support applications with demand performance requirements. The Multipoint Connection-oriented High-performance Internet Protocol (MCHIP) was developed by the authors and is briefly described. Examples are given of ways to provide a connection-oriented (connectionless) internet service on and across connectionless (connection-oriented) subnets. For further details, references are given.
- A generic address scheme to allow one to specify terminal devices that may be located in any of the subnetworks. The proposed form of an address is as follows: OBJ::DOM1::DOM2::DOM3::.... OBJ denotes an object (e.g., terminal) and DOM denotes a domain (e.g., subnetwork, or subnetwork together with the address of some entity within the subnetwork). This format accommodates subnetworks with their own addressing schemes, is not dependent on detailed global knowledge of network topology, and is independent of a device's current network location.

- A parametric description of subnet capabilities and connection requirements including bandwidth allocation, packet loss rates, packet delay, and multipoint capability.

Finally, the authors conclude with a discussion of end-to-end and host interface issues. They argue that there should be a few application-oriented lightweight transport protocols (ALTPs) that can provide variable grade end-to-end flow and error control to different classes of applications. The transport protocols should be simple, designed to be implemented in VLSI and well integrated into the host architecture and operating systems; they should provide reliability and performance guarantees as requested by specific classes of applications. Examples are given of various ALTPs including voice, file transfer, and interprocess communications.

2.6.2 General Comments

This paper provides some good requirements for a network model to support a heterogeneous environment, although I am not sure about the value of having several application-dependent ALTPs at the transport layer. The references are somewhat dated; recent references refer to the authors' work.

2.6.3 Categorization

Internet model (supporting multipoint communications) for high-speed networks in a heterogeneous environment.

2.7 BALA, CIDON, AND SOHRABY, 1990

This paper summarizes the article "Congestion Control for High Speed Packet Networks," by K. Bala, I. Cidon, and K. Sohraby. It appeared in the *Proceedings of IEEE INFOCOM '90*, June 1990, on pages 520 through 526.

2.7.1 Summary

The authors describe a simple implementation of the generalized leaky bucket scheme, using the traditional leaky bucket with the addition of a marker and a spacer, as a packet-level preventive congestion control policy for a high-speed packet switched network. This scheme is recommended for rate regulation of a bursty source.

This paper covers

- General framework of the overall scheme and the methodology
- Details of the generalized leaky bucket scheme for packet-level congestion control
- Control strategies that can be used at each intermediate node to handle congestion.

The authors argue that conventional mechanisms for congestion control within the network based on end-to-end or hop-by-hop windowing schemes are unsuitable for high-speed networks. They suggest a simple open-loop control mechanism that uses knowledge of the extrinsic parameters associated with the connection to control the source by forcing it to conform to these parameters. These schemes are known as input rate regulation schemes; leaky bucket is such a scheme.

Based upon the above arguments, the authors suggest a leaky bucket scheme operating on a session basis that limits the session's average rate and the source's burstiness. This restrictive control is combined with an optimistic bandwidth usage scheme that works by marking packets with two different colors–green and red. The average green packet rate entering the network is the reserved average rate. The average red packet rate represents traffic in excess of this guaranteed average rate and is sent to further utilize unused bandwidth in the network. Both types of packets are further filtered by a spacer that limits the peak rate at which the packets enter the network. The marked packets are then sent into the network where they are treated according to their color by means of a simple threshold policy at each intermediate node.

The following briefly summarizes the basic mechanisms for marking packets, spacing transmission of packets, and thresholding.

- Marking. Each packet is allocated a single color via tokens that depend upon packet lengths. A packet cannot be fragmented and must wait for all the single colored tokens it requires before entering the network. In the absence of enough green tokens, if the admission queue length is smaller than K the packet is forced to wait for green tokens only; otherwise, it looks for enough red tokens.
- Spacing. The spacer introduces a minimum gap between consecutive packet transmissions and limits the peak rate, thereby preventing the flooding of a slow link on the session's path. The space length is a function of the previous packet length and the slowest link on the path.
- Thresholding. Two strategies, based on first-in-first-out service at the intermediate node, were presented.

1. Each node queues at most a threshold T of red packets, after which all subsequent red packet arrivals are discarded and only green packets are accepted.
2. Both green and red packets are queued until the total number of packets equals a threshold T, after which any subsequent red arrivals are discarded.

Statistical analysis led the authors to choose strategy 2. Additionally, they reason that with strategy 1, red packets can occupy queue space without consideration for green traffic. However, strategy 2 stops all reds from entering the queue the moment the queue exceeds T; thus, if the green rate is high, then the greens will occupy most of the queue space and thereby get better service.

Finally, the authors conclude that further work needs to be done to obtain analytical models for their proposed scheme, along with an extensive simulation study to help in determining the parameters of the generalized leaky bucket scheme for different traffic types and grades of service.

2.7.2 General Comments

References are given in the article to additional papers on congestion control for high-speed networks.

2.7.3 Categorization

Proposed scheme for congestion control for high speed packet switched networks: generalized leaky bucket with a marker and spacer.

2.8 CLARK, SHENKER, AND ZHANG, 1992

This paper summarizes the article "Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism," by David D. Clark, Scott Shenker, and Lixia Zhang. The paper appears in the August 1992 *Proceedings of SIGCOMM '92* on pages 14 through 26.

2.8.1 Summary

The authors present an enhanced network architecture for providing support to real-time applications. The motivation for this architecture is based upon the observation that the distinction between telephony networks and computer networks is becoming blurred. Merging the services into a single network offers several advantages, including economies of scale, ubiquity of access and improved statistical multiplexing. This architecture has four key components:

1. The nature of commitments made by the network when it guarantees a certain quality of service
2. The service interface between the source and the network
3. The packet-scheduling algorithm required to meet the service
4. The means by which traffic and service commitments become established, i.e., the admission of new sources.

However, the authors note that this paper does not address all the issues that need to be considered in a final architecture. This paper focuses only on those issues related to the nature of real-time traffic and how those issues affect the four key components.

The initial sections of this paper concentrate on discussing the properties of a particular class of real-time traffic and how delay affects that class. The class of applications chosen for study are those that can be characterized as play-back. The authors believe that most future real-time applications will fit this paradigm. In these applications, the source takes some signal, packetizes it and transmits it over the network while the receiver then depacketizes the data and attempts to play-back the signal. The network, however, introduces some variation in delay, called *jitter*, in each delivered packet. To remove the jitter, the receiver buffers the data and then replays the data at some play-back point. Data arriving after a play-back point is useless.

This particular class of traffic is then subdivided by the authors into two kinds. In one kind, the applications use an *a priori* delay bound advertised by the network to set the play-back point; the other kind adaptively sets the play-back point to one that minimizes delay and provides an acceptable loss rate. Adaptive applications are also noted to be more tolerant, while rigid delay applications are intolerant.

From the categorization presented above, the authors define two kinds of service commitments to meet the traffic requirements. One kind is *guaranteed service* for rigid, intolerant applications; the other is *predicted service* for adaptive, tolerant applications. The authors note that there is a third class of traffic involving datagrams, where no service commitments need to be made. This class of traffic is also known as *best-effort*.

Now that the nature of the commitments is defined, the authors proceed to develop the packet-scheduling algorithm needed to met these commitments. They first present a solution to the guaranteed service commitment, using a token bucket filter and weighted fair queueing (WFQ). They also have chosen WFQ because under the name of packetized generalized processor sharing

(PGPS), Parekh and Gallager have proven that, under certain conditions, this algorithm can deliver a guaranteed quality of service. A brief technical description is then presented by the authors but we will only summarize the key ideas below.

First, to characterize a flow, the authors have chosen a token bucket filter. This filter is characterized by two parameters, a rate r and a depth b, where r is the rate at which tokens are being deposited and b being the bucket's maximal depth. A traffic flow conforms to a token bucket filter (r,b) if there are always enough tokens in the bucket whenever a packet is generated.

Now consider some set of flows and a set of clock rates r^{α}. The clock rate of a flow represents the proportion of the total link bandwidth which this flow will receive when it is active. By assigning it a clock rate r^{α}, the network commits to providing this flow an effective throughput rate no worse than $(\mu\alpha)/(\Sigma_{\beta}r^{\beta})$, where μ is the link speed and the sum in the denominator is over all active flows. Parekh and Gallager have further shown the result that, in a network with arbitrary topology, if a flow gets the same clock rate at every switch and the sum of the clock rates of all the flows at every switch is no greater than the link speed, then the queueing delay of that flow is bounded above by $b^{\alpha}(r^{\alpha})/r^{\alpha}$. This flow rate is *independent* of the other flows' characteristics.

The flow formulation above can be made precise in the context of a fluid flow model of the network where the bits drain continuously out of the queues. We present only their packetized version, though the fluid model is presented first in the paper. Define $\delta_i^{\alpha}(t)$ for all $t \geq t_i^{\alpha}$ as the number of bits that have been serviced from the flow α between the times t_i^{α} and t. Associate with each packet the function $E_i^{\alpha}(t) = (m^{\alpha}(t_i^{\alpha} - \delta_i^{\alpha}(t))/r^{\alpha}$ where we take the right-hand limit of m; this number is the level of backlog ahead of the packet i in the flow α's queue divided by the flow's share of the link and can be thought of as an *expected* delay until departure for the last bit in the packet. The algorithm is at any time t; when the next packet to be transmitted must be chosen, select the packet with the minimal $E_i^{\alpha}(t)$.

The authors then present an intuitive argument to show why WFQ is not a good algorithm for predicted service. Its emphasis on isolation prevents the effective sharing of resources, especially in times of bursty traffic. This is because a WFQ algorithm would continue to send packets at their clock rates. The goal of predicted service is to schedule packets so that they achieve the lowest delay bounds. The play-back problem is analogous to deadline scheduling; and for this class of scheduling, the deadline-first scheduling algorithm has been proven optimal. Thus, the authors examine FIFO queueing as a better discipline for predicted service than WFQ. The authors use simulation results to verify their argument.

However, the authors note that if you need to traverse several hops, the jitter tends to increase dramatically because of the opportunity for uncorrelated queueing delays. To avoid this problem, the authors propose to use FIFO+ as the queueing discipline. In this queueing technique, packets are ordered by their expected arrival time. Again, the authors use a simulation to verify that FIFO+ achieves a slower rate of growth in delays as multiple hops are traversed, when compared to WFQ or FIFO queueing.

After presenting each scheduling algorithm, the authors then combine them to provide service for guaranteed, predicted, and datagram traffic. They argue that since they must isolate the guaranteed flow from the other two flows to meet the service commitments, a time-based WFQ scheme must be used as a framework into which the algorithms are merged. Each guaranteed service client α has a separate WFQ with some clock rate r^{α}. The predicted and datagram services are then assigned to a pseudo WFQ flow, known as 0, with on each link $r^0 = \mu - \Sigma_{\alpha}r^{\alpha}$, where the sum is over all the guaranteed service flows passing through that link. Within this flow 0, there are a number of strict priority classes, and within each class the FIFO+ algorithm is used. The effect of priority is to shift the jitter of a higher priority class to a lower class. Datagram service is therefore assigned to the lowest priority class.

Now that the nature of the commitments have been defined and the packet scheduling algorithm presented, the authors move on to define the service interface each algorithm requires. For guaranteed service, only the clock rate r^{α} is needed. For predicted service, thc service interface must characterize both the traffic and the service. This, therefore, includes the filter rate on size (r,b) and the delay and loss characteristics. To be able to provide predicted service, enforcement must be carried out. However, the authors note it is only necessary to check conformance at the edge of the network, since any later violation would be due to the scheduling policies and load on the network.

The final component of the architecture, the admission control procedure, is discussed. The paper does not address the negotiation process but instead concentrates on the conditions under which a network will accept or reject an admission. First, they arbitrarily select an upper bound of 90 percent for real time traffic. This bound ensures that datagram service continues at all times; furthermore, the authors believe that it will provide enough spare capacity to accommodate most fluctuations in the other service classes. The authors then present an illustration of the considerations involved in designing an admission control policy, but do not present a specific definition. This definition is the focus of their current work.

The authors admit that there are other service features that may need to be supported but are not covered in the context of the scheme presented here. These include

- The traffic priorities within a single stream, so a source can control which packets get dropped
- Dropping packets within the network, when the network knows the packets will not meet their deadlines
- Buffering packets within the network so that they do not arrive early at the destination.

Other researchers are also addressing the problems created by real-time applications in a packet-switched environment. Related work includes

- Delay-EDD
- MARS
- Jitter-EDD
- Hierarchical Round Robin
- Statistical-EDD
- Stop and Go Queueing.

WFQ, Delay-EDD, and MARS are work-conserving scheduling algorithms, in that nothing is left idle if there is a packet in the queue; while stop and go queueing, hierarchical round robin, and Jitter-EDD are non-work-conserving. The authors note, however, that none of these algorithms deal with both guaranteed and predicted services. Most algorithms address only the needs of rigid and intolerant applications, which they believe will not be the service requirements of most real-time applications.

2.8.2 General Comments

The authors present many interesting ideas on the subject of how to support real-time applications. The terms are well-defined and the motivation for each step is clear. However, as the acknowledgment admits, this is an attempt to clarify ideas, so that at times the paper looses focus and seems to go off on tangents. An extensive reference list is also provided.

2.8.3 Categorization

Architecture and algorithms for providing support to real-time "play-back" applications requiring guaranteed and predicted service.

2.9 FLOYD AND JACOBSON, 1992

This paper summarizes the article "Random Early Detection Gateways for Congestion Avoidance," by S. Floyd and V. Jacobson. This article is an advance draft copy, dated 2 September 1992, and as such, has not been published in any journal or presented at any conference.

2.9.1 Summary

This paper describes a gateway algorithm, Random Early Detection (RED) for congestion avoidance in packet-switched networks with window-based flow control. The RED algorithm measures and controls the average queue size. Incipient congestion is detected when the average queue size exceeds a minimum threshold. Transient congestion is accommodated by a temporary increase in the queue. Longer-lived congestion is reflected by an increase in the average queue size, and results in feedback to some of the connections to decrease their windows.

The authors first describe their algorithm, including design goals and guidelines, and optimal settings for the algorithm parameters. A comparison of RED with previous work on congestion avoidance gateways is then provided, followed by an evaluation of RED, which is supported by simulations. The article concludes with implementation issues and further studies.

The basic algorithm is as follows. The RED gateway calculates the average queue size, using a low-pass filter that calculates an exponential weighted moving average. When the average queue size is less than the minimum threshold, no packets are marked. When the average queue is greater than the maximum threshold, every arriving packet is marked. If marked packets are in fact dropped, or if all source nodes are cooperative, this ensures that the average queue size does not significantly exceed the maximum threshold. When the average queue size is between the minimum and maximum threshold, each arriving packet is dropped with probability p_a, where p_a is a function of the average queue size. Each time that a packet is marked, the probability that the packet is marked from a particular connection is roughly proportional to that connection's share of the throughput at the gateway. The RED gateway has two separate algorithms. The algorithm for computing the average queue size determines the degree of burstiness that will be allowed in the gateway queue. The algorithm for calculating the marking probability determines how frequently the gateway marks packets, given the current level of congestion. The goal is for the gateway to mark packets at fairly evenly spaced intervals and to mark packets sufficiently frequently to control the average queue size.

Thus, the main goals and design guidelines of RED are

- *Avoiding bias against bursty traffic*
- *Utilizing distinct algorithms for congestion detection and for feedback*, which helps to avoid performance biases such as biases due to traffic phase effects and biases against bursty traffic
- *Avoiding global synchronization*, which normally results from notifying all connections to reduce their windows at the same time
- *Randomly selecting connections to notify*, which avoids global synchronization, traffic phase effects, and the bias against bursty traffic
- *Responding quickly to high congestion* by marking packets frequently when congestion is high

- *Allowing for a choice of feedback mechanisms* for notifying a source node to reduce the congestion control window for that connection
- *Maintaining an upper bound on the average queue size*, which allows RED to control the average queue size
- *Maintaining simplicity in algorithm design.*

Before describing simulation results, the authors then discuss earlier work on congestion avoidance gateways and, in particular, compare RED with Random Drop, Early Random Drop, Drop Tail, and DECbit gateways. The authors argue that these gateways have their drawbacks. For instance, Drop Tail introduces global synchronization, resulting in a loss of throughput at the gateway; Random Drop and Early Random Drop do not successfully control misbehaved users (the authors show that RED can be easily modified to detect misbehaved users); Early Random Drop drops packets with a fix probability which is independent of network traffic; Drop Tail and Random Drop have a bias against bursty traffic; and DECbit exhibits traffic phase effects and a bias against bursty traffic.

The RED algorithm has been characterized and compared to Random Drop, Early Random Drop, Drop Tail and DECbit gateways through simulations. Simulation results have verified that RED has effectively met the design goals and guidelines listed above, in addition to those of fairness, low parameter sensitivity, and appropriateness for a wide range of environments.

Through simulations and theoretical analysis, the authors provide a few rules to ensure adequate performance of their algorithm:

- To ensure adequate calculation of the average queue size, set the queue weight factor to greater than or equal to 0.001.
- To maximize network power, set the minimum and maximum threshold for the queue sufficiently high.
- To avoid global synchronization, make the difference between the maximum and minimum threshold for the queue sufficiently large.

The authors conclude by describing further work in this area, including determining the optimum average queue size for maximizing throughput and minimizing delay for various network configurations; the handling of misbehaved users; implementation efficiency; determining whether the queue should be measured in packets or in bytes; and the servicing of queues.

2.9.2 General Comments

This paper is generally well written (however, since it is a draft, the reviewer feels that some reordering of the sections is needed) and provides a good balance between design goals, the algorithm and its parameter settings, implementation, simulation results, and issues for further investigation.

2.9.3 Categorization

Gateway congestion avoidance algorithm for packet-switched networks with window-based flow control.

2.10 GOLESTANI, 1990

This paper summarizes "Congestion-Free Transmission of Real-Time Traffic in Packet Networks," by S. Jamaloddin Golestani. The article appeared in Volume II of the June 1990 *Proceedings of IEEE INFOCOM '90* on pages 527 through 536.

2.10.1 Summary

This paper describes a framing strategy for congestion-free communication in packet networks, applicable to high-speed network environments with a diverse mix of traffic types and service requirements. The strategy is composed of two parts: a packet admission policy imposed per connection at the edges of the network, and a particular queuing scheme practiced at the switching nodes, which is called stop-and-go queuing. This strategy for congestion control provides guaranteed services per connection with no packet loss and a constant end-to-end delay, plus a small bounded jitter term. The author first describes the problems associated with control of congestion and improvement of delay performance in packet networks, then focuses on the specific issue of the formation of long bursts inside a packet network, which increases the chance of buffer overflow. This issue motivates the author's congestion control strategy, which is then described, along with delay performance considerations and strategy tradeoffs. The paper concludes with a discussion of implementation of stop-and-go queuing.

The author's approach is based on his observation of the formation of packet bursts inside a network. He shows that despite perfect regulation and smoothing of arrived traffic at a network's edge, the network is capable of creating long bursts of traffic, causing loss and delay. These bursts result from the complicated way in which different traffic streams interact with each other. He argues that more elaborate controls are necessary to provide guaranteed services with stringent loss and delay requirements for time-critical applications in a packet network.

The author's framing strategy eliminates the packet clustering effect inside the network and guarantees that once the packet streams are smooth, the smoothness is preserved throughout the network. This property makes the worst-case queue sizes predictable and permits an environment for congestion-free communication.

2.10.1.1 Framing Strategy

The following summarizes the author's framing strategy by defining the admission policy and stop-and-go queuing scheme.

- The admission policy is based on the definition of smoothness. Once a connection k is set up in the network and a transmission rate r_k is assigned to it, its packet arrival to the network is required to be (r_k,T)-smooth. This means that for a connection k that uses an access link l, the total length of admissible packets during each arriving frame of l is limited to r_kT bits. T is the time interval length of a frame.
- The stop-and-go-queuing scheme is based upon two rules:
  - Rule 1: Consider a node n, an incoming link l', an outgoing link l, and a packet that has arrived during a frame F' of link l' and seeks service by link l. Let F be the departing frame of link l that is adjacent to the arriving frame F'. Transmission of the packet should not start before the beginning of frame F.

- Rule 2: A link should not stay idle while there is an eligible packet left in the queue. A packet is eligible at time t if it can be offered service at time t without violating Rule 1. A packet is designated eligible only at the beginning of departing frames, and during any departing frame no new packet is added to the eligible job.

Given the admission policy and stop-and-go queuing scheme, the author concludes that (1) at any queue, once a packet is marked as eligible for transmission, it will receive service within T seconds; (2) the packet stream of each connection will maintain the original smoothness property throughout the network; and (3) a buffer space of at most $3C_lT$ per link l is enough to eliminate buffer overflow in the network (C_l is link capacity.)

2.10.1.2 Issues

The following briefly highlights issues discussed in the paper that relate to the framing strategy.

- Given the above framing strategy, the author discusses delay performance. He shows that end-to-end delay of packets of a connection is constant, except for a delay jitter between $-T$ and T. Although packet delay with conventional FIFO queuing can be smaller on the average than that of the framing strategy, the delay is distributed over a wide range, which makes the behavior of the network relatively unpredictable and subject to buffer overflow and packet loss. He concludes that delay performance resulting from his strategy is attractive, given that the total queuing delay of the connection is acceptable.
- The benefits of the framing strategy are basically obtained by incurring two performance costs: the extra delay of the stop-and-go queuing, and the strict admission policy of enforcing smoothness on packet arrivals. The queuing delay can be engineered to be within the acceptable range by appropriate choice of the frame size T. The author suggests that it may be desirable to incorporate multiple frame sizes into his strategy. The real cost of the framing strategy is thus due to the policy of admitting a limited number of packets per connection in each frame. The author recommends that less conservative traffic control and management strategies should be combined with the framing strategy.
- The author also discusses the implementation of the framing strategy and concludes that implementation is simple, with little processing overhead and minor hardware modifications to the conventional FIFO queuing structure.

2.10.2 General Comments

This is a well written paper on the author's simple strategy for congestion control, presenting his ideas and reasons in a straightforward manner. Further details on stop-and-go queuing can be found in "A Stop-and-Go Queuing Framework for Congestion Management" by S. J. Golestani, *Proceedings of SIGCOMM '90*, pages 8 through 18.

2.10.3 Categorization

Congestion control strategy for packet networks.

2.11 LAI, 1990

This paper summarizes the article, "Protocols for High-Speed Networking," by W. Lai. It appears in Volume II of the June 1990 *Proceedings of IEEE INFOCOM '90* on pages 1268 and 1269.

2.11.1 Summary

This paper focuses on approaches for increasing protocol speeds. The author begins by describing bottleneck problems for wide-area networks and protocol processing and then discusses three approaches to increase protocol speeds. He concludes by briefly describing new design issues in high-speed networks and various current activities in this area.

The author begins by describing how the increase in processing speeds of communications processors has not been keeping up with the increase in transmission bandwidth; thus, the bottleneck has now shifted to processing of packets in network nodes and end systems. Based on these assumptions, three approaches to increase protocol speeds are described. Following are highlights of the author's main points.

- Improved implementation of protocols
  - Protocols should be properly embedded in their operating system environment, and a low-overhead process structure should be available for protocol processing.
  - Overhead can be reduced by the optimization of the execution path for normal data handling, the use of bigger packet sizes, the use of packet groups (i.e., for acknowledgments and control), and the efficient control of protocol dynamics.
  - Hardware assistance, such as pipelining and parallel processing or custom VLSI, can be used.
- Packet formatting optimization: Formatting principles to streamline packet processing include alignment (i.e., word boundaries); appropriate placement of protocol control information in header and/or trailer; single use of a control field; and a fixed format for different types of packets.
- Use of success-oriented protocol architectures
  - Network functions can be streamlined by providing only a common core set of functions that apply to all information flows.
  - End-system functions can be streamlined via lightweight protocols that are tailored to particular end-to-end communication needs.

The discussion then switches briefly to new design issues. With high-speed networks, round-trip delays will primarily be caused by propagation delays and will remain constant. Thus, the higher the transmission speed, the more bits there are in the pipe. To cope with the high volume of traffic, algorithms such as selective retransmission, and new control mechanisms, are needed.

Finally, current activities in high-speed networking are listed. These include work on a header prediction algorithm, NETBLT, VMTP, frame relay for ISDN, ATM for B-ISDN, and work by several ANSI task groups.

2.11.2 General Comments

This article is very brief and nontechnical: it provides a couple of good references.

2.11.3 Categorization

Proposals for increasing protocol speeds for high-speed networks.

2.12 MITRA, 1990

This paper summarizes the article "Optimal Design of Windows for High Speed Packet Networks," by D. Mitra. It appears in Volume III of the June 1990 *Proceedings of IEEE INFOCOM '90* on pages 1156 through 1163.

2.12.1 Summary

In this paper, the author examines the basic mechanism of sliding windows for the congestion control of virtual circuits. The problem of optimal design of windows is formulated and solved; formulas are derived for basic quantities such as throughput, delay, and packet queue lengths.

The author's work is based on the central thesis that the combination of the propagation delays that exist in geographically dispersed data networks and the high transmission rates of the emerging networks call for completely new design procedures. As the transmission speeds scale up, the propagation delays, which do not scale, become increasingly influential in determining performance. This is the motivation behind the asymptotic theory of the paper. The development of asymptotics is distinct for the three regimes of light, moderate, and heavy nodal usages. For a broad spectrum of performance functionals, the asymptotically optimal regime is the middle one, on which the analysis focuses (i.e., nodal processing rate).

It is demonstrated that existing design rules based on neglecting the effects of propagation delay yield substantially worse performance, albeit with much lower memory requirements than the rules proposed by the author. The following summarizes the author's efforts:

- Adaptive, dynamic windowing. A design equation is given that relates the mean response time to the optimal window size. This equation is used to adjust the window, based on measurements of the response time in a possibly nonstationary environment. The optimal window size is λ, where λ is the mean number of packets of the virtual circuit that may be processed and transmitted at a single node in an interval equal to the mean propagation delay.
- Buffer sizing. An "optimistic buffer design" approach is proposed in which buffer sizes are a small multiple of $\sqrt{\lambda}$, rather than $o(\lambda)$ as in the standard conservative approach. One of the reasons behind this optimistic approach is that the overwhelming majority of packets in an optimum window are found not in the nodal buffers, but in the transmission pipeline. Although the use of small buffers leads to buffer overflows and consequent retransmissions, with proper design, overflows are rare and the degradation in performance due to retransmissions is small.
- Multiple virtual circuits. The memory requirements for an implementation in which each virtual circuit has a dedicated buffer are compared to those of another in which a shared buffer is used.

2.12.2 General Comments

This paper is very theoretical. An even more detailed description of the author's work can be found in "Optimal Design of Congestion Control for High Speed Data Networks," AT&T Technical Report, September 1989. For completeness and continuity, some of the results of this paper are included in the summary above. The AT&T technical report "A Go-Back-n Protocol

Which is Efficient in High Speed Data Networks with Small Buffers," by D. Mitra and I. Mitrani, January 1990, demonstrates the effectiveness of the results of the previous two papers by using the go-back-n protocol for data retransmissions.

2.12.3 Categorization

Examination of the basic mechanism of sliding windows for congestion control, including queue buffer sizing.

2.13 RAMAKRISHNAN AND JAIN, 1988

This paper summarizes the article "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with a Connectionless Network Layer," by K. Ramakrishnan and R. Jain. It appeared in the *Proceedings of SIGCOMM '88*, August 1988, on pages 303 through 313.

2.13.1 Summary

This paper describes the Explicit Binary Feedback Scheme, which uses a connectionless network layer protocol for congestion avoidance in networks. Briefly, the scheme uses minimal feedback (one bit in each packet) from the network to adjust the amount of traffic allowed into the network. Each congested network server (router/link) sets the congestion avoidance bit. This congestion indication is returned to the network user via transport level acknowledgment and is then utilized by the network user to control information flow into the network. The scheme is distributed, adapts to the dynamic state of the network, converges to the optimal operating point, is simple to implement, has low overhead, and maintains fairness.

The authors first summarize their feedback scheme, which consists of a router policy and a user policy. The overall model is then presented, including definitions for efficiency and fairness. This is followed by a description of each of the policy's sub-elements, including the analysis that went into the choice of the various mechanisms based upon the model. The paper concludes with a discussion of the behavior of the scheme with random packet-size distributions and under transients.

The following summarizes the authors' scheme by defining the router and user policies and also lists the issues/topics addressed in the paper.

- Router Policy
  - Congestion Detection. The router sets the congestion avoidance bit in the packet when the average queue length at the router at the time the packet arrives is greater than or equal to one. Issues discussed include router utilization versus queue length to detect congestion and hysteresis versus threshold policy for generating the feedback signal (setting the congestion avoidance bit).
  - Feedback Filter. A filter is needed to pass only those states of the routers that are expected to last long enough for the user action to be meaningful. Thus, determination of the average queue length is based on the number of packets in the network router that are queued and in service, averaged over an interval *T*. This interval *T* is the last cycle time plus the busy period of the current cycle. A cycle is defined as *busy+idle time*. Issues discussed include weighted, exponentially running average queue size; fixed vs. dynamic time intervals; and the use of the current cycle for determining average queue size.
- User Policy
  - Decision Frequency. The user updates the window size after receiving acknowledgments for a number of packets transmitted. This number is the sum of the previous window size (W_p) and the current window size (W_c) at which the transport connection is operating. The bits returned in the acknowledgment are

stored by the user. Issues discussed include altering the window size after every acknowledgment, window oscillation, and feedback delay after window size change.

- Use of Received Information. The only bits examined are those that correspond to the last W_c packets for which acknowledgments are returned. Issues discussed include the use of information prior to window change.
- Signal Filtering. If at least 50% of the bits examined are set, the window size is reduced from its current value of W_c. Otherwise, the window size is increased. Issues discussed include multiple vs. single cut-off factors and the dependency of the cut-off factor on the policy used by the routers to detect congestion.
- Decision Function. When the window size is increased, W_c is incremented by 1. When the window size is decreased, it is decreased to $0.875*W_c$. Issues discussed include additive vs. multiplicative increase/decrease, window oscillations, and efficiency.

Finally, the behavior of the binary scheme was tested under various transients and pathological changes. Results include the following:

- The multiplicative decrease/additive increase algorithms show that two sources with randomly distributed packet size distributions reach a fair value of the network resources allocated to them.
- After transients in service time (resulting from changes in the network) occur, the overall window size recovers and the network operates at its new maximally efficient point.
- Two users who start at different times in the network converge to a fair value so that their window sizes are nearly equal.

2.13.2 General Comments

This article presents the authors' algorithm clearly, providing motivation for the model and justifying the parameter choices for optimization of the algorithm.

2.13.3 Categorization

Congestion avoidance algorithm (feedback scheme) for networks using connectionless network layer protocols.

2.14 SHACHAM AND MCKENNEY, 1990

This paper summarizes the article "Packet Recovery in High Speed Networks Using Coding and Buffer Management," by N. Shacham and P. McKenney. It appeared in Volume I of the June 1990 *Proceedings of IEEE INFOCOM '90* on pages 124 through 131.

2.14.1 Summary

This paper presents a unique method based on forward error correction (FEC), which allows the destination to reconstruct missing data packets by using redundant parity packets that the source adds to each block of data packets. The coding technique is suitable for use in gigabit, wide-area networks (GWANs), reducing the need for retransmissions of reliable data, enhancing the quality of real-time data that cannot rely on acknowledgments and retransmissions, requiring only small data storage for operation, and being amenable to hardware implementation. The authors discuss (1) coding for packet recovery, (2) buffer management and interleaving, and (3) performance evaluation. The following briefly summarizes the authors' ideas, with emphasis on their performance results.

2.14.1.1 Coding

The authors discuss the motivation for their design of the coding scheme, present several schemes suitable for a GWAN environment, and consider the special case in which erasures occur in packet-long sequences. Briefly, their approach utilizes sequence numbers, which are already required by many protocols. Since packets are sent in increasing order, a data recipient identifies missing packets (sequence of bit erasures) by gaps in the arriving sequence. The algorithm groups data packets into blocks of a predetermined size, and adds to each block a number of parity packets to contain the error-control bits. The number of parity packets, and their construction, determines the maximum number of data packets that can be recovered. A *vertical* parity packet scheme is described for single packet recovery, and a *diagonal* parity packet scheme is described for multiple packet recovery. These coding and encoding schemes add minimal delay to the packets and require only small data storage for operation.

The authors also extend their discussion to handling of bit errors and recovering bursts of lost packets.

2.14.1.2 Buffer Management and Interleaving

To alleviate the effect of packet loss correlation, the authors' approach also considers (1) buffer management procedures in the networks that would reject packets based on their block affiliation and the number of packets already lost in their block; and (2) interleaving the data either intentionally (deterministically) by the source, or as it occurs naturally during statistical multiplexing. Interleaving helps to spread bursts of deleted packets in the arriving stream over multiple blocks.

2.14.1.3 Performance Evaluation

Both an analytic model and more realistic simulations were used to show the limitations of the coding schemes and the effects of buffer management and interleaving. Packet loss ratio (ratio of packet loss rate after decoding to the rate when no coding is used) was used as the performance measure. Simulations showed that a significant reduction in packet loss rate was achieved with a combination of coding, buffer management, and interleaving. Some important findings are as follows:

- Buffer management shows a total improvement of three orders of magnitude (analytical results).
- Packet loss correlation is a severe problem when each packet stream has a dedicated finite buffer. Buffer management improves the loss ratio by up to two orders of magnitude (simulation results).
- Deterministic interleaving achieves better loss ratios than statistical multiplexing. Loss ratios of better than 10^{-3} were demonstrated (simulation results).

2.14.2 General Comments

This is a good paper describing a novel technique.

2.14.3 Categorization

Packet recovery algorithm for use in high-speed networks; effective for network congestion avoidance.

2.15 WILLIAMSON AND CHERITON, 1991

This paper summarizes the article "Loss-Load Curves: Support for Rate-Based Congestion Control in High-Speed Datagram Networks," by Carey L. Williamson and David R. Cheriton. It appeared in the *Proceedings of SIGCOMM '91*, September 1991, on pages 17 through 28.

2.15.1 Summary

This paper describes an approach that uses loss-load curves for rate-based congestion control in high-speed datagram networks. Loss-load curves characterize the packet delivery service to the network. In particular, they express the probability of packet loss as a function of offered load at a given time, or of load condition on the network. The network (i.e., gateways) explicitly feeds this information back to network clients (i.e., source hosts) who, depending upon the application, choose their own rate at which to transmit data, based upon tradeoffs between throughput and packet loss. Gateways provide periodically updated curves to their clients, based upon dynamic network conditions (e.g., network capacity).

In their paper, the authors describe the basic loss-load model and illustrate their approach by presenting and evaluating (via analytic and simulation results) a specific loss-load algorithm. A discussion of refinements and extensions of the basic loss-load model is also provided.

2.15.1.1 Loss-Load Model

The loss-load model determines a packet loss probability p_i for each sender such that the excess traffic is discarded. Mathematically expressed, this requirement is $\sum_{i=1}^{M} p_i r_i = r_{total} - C$, where C is the gateway capacity, r_i is the average rate each sender is contributing, and r_{total} is the total load at the gateway. Each packet from sender i is discarded, at random, with probability p_i, to statistically enforce the desired rate of packet loss. The rules used to compute the probabilities are specified by the network manager. Once the rule is specified, the gateway computes this loss-load information as feedback to the source hosts. This information is exchanged using a minimal number of parameters. The host is then responsible for choosing its own operating point on the loss-load curve and multiplexing this rate among all packets. The choice of rate is based on the type of service required by the packets, throughput, the estimated round trip time to each destination, and other factors.

The authors propose a specific formula for the loss-load curve, which will not be detailed here. The formula was designed with three major objectives: (1) keeping offered load close to capacity at all times, (2) encouraging sender cooperation, and (3) providing protection from misbehaving hosts. Using the formula proposed by the authors, only two parameters are fed back from the gateway to the host in order to calculate the loss-load curves of the given gateway. One parameter measures the total (excess) traffic load, and the other measures the distribution of the total load.

2.15.1.2 Analytic Results

Analytic results of the proposed model show several desirable characteristics for a congestion control strategy and include

- Bounded packet loss. Packet loss probability for a cooperating sender is bounded and predictable.

- Responsiveness to load changes. Cooperation from all senders is encouraged and protection is provided from greedy senders. As a sender increases its transmission rate (towards its own maximum rate), other hosts are encouraged to slow down. However, a sender increasing its rate beyond its optimal rate encourages other hosts to speed up.
- Convergence. The network as a whole converges to a stable equilibrium operating point with each sender at its own maximum transmission rate. Convergence is always above the equilibrium load and no oscillation occurs.

2.15.1.3 Simulation Results

Simulation results of the proposed model show additional characteristics of the congestion control strategy. These include

- Responsiveness to load changes. The system returns to equilibrium after load changes, both increases and decreases. Convergence is fast and no oscillations occur.
- Responsiveness to changes in capacity. Since capacity information is implicitly contained in the loss-load curve, senders are able to quickly and efficiently adapt to changes in network capacity.
- Fairness. Regardless of round trip delay, each sender receives the same equilibrium throughput.

2.15.1.4 Refinements and Extensions

The authors conclude their paper by discussing refinements and extensions to their basic model. Topics discussed include the following.

- Controlling queue size at the gateway by discarding $r_{total} - C+\varepsilon$ packets
- Comparison of various queuing disciplines (FCFS, priority, and fair queuing)
- Consideration of delayed resolution, which defers a decision to discard a packet until the end of a service interval (as opposed to the time when a packet arrives)
- Use of type of service to bias packet loss
- Modification of loss-load strategy to perform congestion *avoidance.*

2.15.2 General Comments

This paper presents the authors' congestion control strategy clearly, by first describing the basic model and then the specific proposed formulas. A nice explanation of analytic and simulation results is given and includes comparison to other approaches.

2.15.3 Categorization

Network service characterization to support rate-based congestion control in high-speed datagram networks.

2.16 ZHANG, 1990

This paper summarizes the article "VirtualClock: A New Traffic Control Algorithm for Packet Switching Networks" by L. Zhang. It appears in the September 1990 *Proceedings of SIGCOMM '90* on pages 19 through 29.

2.16.1 Summary

This paper describes an algorithm, VirtualClock, for data traffic control in high-speed networks. VirtualClock maintains the statistical multiplexing flexibility of packet switching while ensuring each data flow its reserved average throughput rate. The author first describes the algorithm, including design goals and modifications made to the initial algorithm that resulted in more robust control. Simulation results are provided that show the strengths of the algorithm. Finally, issues for further investigation are described.

VirtualClock is a rate-based traffic control algorithm. One of the difficult issues is how to monitor and control the transmission rate of statistical data flows, and how to enforce network resource usage to prevent interference among different users without sacrificing the flexibility of statistical multiplexing; VirtualClock solves these problems. It was designed to provide the following functionalities:

- Support diverse throughput requirements from various applications by enforcing the resource usage according to each flow's average throughput reservation
- Monitor average data flows and provide measurement input to other network control functions
- Provide fire walls between individual data flows
- Preserve the full flexibility of statistical multiplexing of packet switching.

The basic idea of VirtualClock was inspired by the Time Division Multiplexing (TDM) system. The author wanted to provide the fire walls of a TDM system, yet preserve the flexibility of statistical multiplexing of packet switching. A TDM system is driven by a real-time clock; the author suggests that a statistical multiplexing system may use a virtual clock concept in a similar way.

Briefly, the algorithm is as follows. Each data flow is assigned a *Virtual Clock* that ticks (i.e., advances) at every packet arrival from that flow; a tick step is equal to the mean interpacket gap. Thus, the VirtualClock reading indicates the expected arrival time of the packet. If a flow sends packets according to its specified average rate, its VirtualClock reading should be near real time. To imitate the transmission ordering of a TDM system, each switch stamps packets by the flows' VirtualClock reading and uses the stamp to order transmissions while preserving the statistical nature of packet switching. The algorithm can thus monitor each flow by comparing its VirtualClock with the real-time clock periodically, in order to provide feedback to flow sources if their actual throughput ever departs significantly from the reserved rate. The author details the VirtualClock as a data flow monitor and describes other features such as priority service and building fire walls (avoiding interference among flows).

The author further describes revisions that have been made to the algorithm as a result of simulations. These revisions addressed issues such as these: (1) a flow cannot increase the priority of its packets by saving credits within the average interval (AI is the interval over which a flow's VirtualClock is checked); and (2) a flow source must constrain itself from sending more than the allowed number of packets during each interval (this is the user-behavior envelope).

The VirtualClock algorithm has been tested extensively through simulations. The following summarizes the test results for the model used:

- Flows with same throughput requirement
  - The network meets the flows' average throughput requirement.
  - The average queueing delay is low.
  - The network load is stable and congestion free.
  - The network provides a fair service, independent of flows' path lengths.
- Supporting diverse flow throughput
  - The user's expected throughput is satisfied; different path lengths show no effect.
  - Lower throughput flows seem to experience a higher queueing delay.
- Building fire walls between flows
  - Normal flows are well protected from (the few) misbehaving users.
  - When the misbehaving users drove the link utilization to 100%, queueing delay of the normal flows remained about the same (as with the previous experiments).

The author concludes by describing further work in this area, including how to design application protocols that can automatically adjust to constraints such as the user-behavior envelope, performance of VirtualClock under highly bursty traffic, resource overbooking versus delay reduction, and fine-tuning of the average interval.

2.16.2 General Comments

This paper is well written. The algorithm is simple (although this reviewer needed to reread a page or two, to figure it out); the author first describes the initial version of the algorithm and then discusses the important issues that resulted in the final version. Simulation results are well summarized so as not to overwhelm the reader and are to the point.

2.16.3 Categorization

Data traffic control algorithm for use in high-speed networks.

2.17 ZHANG, DEERING, ESTRIN, SHENKER, AND ZAPPALA, 1993

This paper summarizes the article "RSVP: A New Resource ReSerVation Protocol" by L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. It is a draft copy of a submission to SIGCOMM '93.

2.17.1 Summary

This paper presents a reservation protocol for creating and maintaining resource reservations for both unicast and multicast applications. The authors motivate the development of this protocol by pointing out the inadequacies of the current point-to-point best effort service for the new classes of distributed applications being developed. They also note that their protocol is only part of a larger architecture necessary to support the new application requirements. The other pieces of the architecture are

- A flow specification, or *flowspec*, which describes the characteristics of both the traffic stream sent by the source and the service requirements of the application
- A routing protocol that can provide quality unicast and multicast paths
- An admission control algorithm that can maintain the network load at a proper level
- A packet scheduling algorithm that can provide the committed qualities of service, once the flow has been accepted.

Specifically, the authors present their design goals, the design principles used to meet these goals, a detailed description of the protocol operation, a simple example, the current state of the implementation, and related and future work.

The authors determine their design goals by first demonstrating that simple extensions to existing protocols are inadequate for point-to-point, point-to-multipoint, and multipoint-to-multipoint reservations. In particular, the obvious extensions do not address the properties of multiple, heterogeneous receivers and/or multiple senders. In the process of defining the inadequacies, the authors identified the following seven important design goals:

1. Accommodate heterogeneous receivers.
2. Adapt to changing multicast group membership.
3. Exploit the resource needs of different applications in order to use network resources efficiently.
4. Allow receivers to *switch channels*: i.e., the receiver should have the ability to control which packets are carried on its reserved resources.
5. Adapt to changes in the underlying unicast and multicast routes, so that the resource reservations are automatically reestablished along the new paths as long as adequate resources are available.
6. Control protocol overhead, so that it does not grow linearly (or worse) with the number of participants.
7. Make the design modular, to accommodate heterogeneous underlying technologies.

The authors stress that RSVP is the application's vehicle to communicate their requirements to the network. RSVP delivers these requests to the switches but is not responsible for delivering the services. The kinds of services RSVP can support are those that only rely on state being established at the individual switches along the paths that are determined by the routing algorithm.

The authors then define and elaborate upon six basic design principles to achieve the design goals presented above. These principles are summarized below, along with the benefits.

Receiver-initiated resèrvation is unique in letting the receivers choose the level of resources reserved and be responsible for initiating and keeping the reservation active. The receiver takes the primary role because the receiver is the one who knows its own limitations and is the one affected by the quality of service received.

Separating the reservation from packet filters enables the capability of switching channels. A resource reservation merely assigns certain resources to the entity making the reservation. The entity making the reservation is responsible for controlling which packets can use those resource via a filter.

Different reservation styles, in the form of filters, can be one of three types:

- No filter
- Fixed filter
- Dynamic filter.

No-filter reservation implies that any packets destined for a particular multicast group can use the reserved resources. A fixed-filter reservation means that the receiver will receive data only from the sources listed in the original reservation request for the duration of the reservation. A dynamic-filter reservation allows the receiver to change its filter to different resources over time. These different styles allow the individual switches to decide how the individual reservation requests can be merged.

Maintaining soft-state at the switches allows the ability to adjust to dynamic changes in the network. The soft-state information consists of path and reservation state. Each source sends a path message that establishes or updates the path's state, while the receiver sends a reservation message that establishes or updates the reservation. The path messages follow the routing decision of the routing protocol and carry a flowspec as well as a flag indicating if filtered reservations are allowed. The reservation message contains a flowspec, a reservation style and a packet filter if the reservation style is a fixed-filter or a dynamic-filter. Both message types also carry a timeout value used by the switches to delete the state. It is the responsibility of the senders and receivers to prevent timeouts by refreshing the state periodically.

Controlling protocol overhead is a key concern to achieve efficient scaling. The RSVP overhead is determined by three factors:

1. The number of RSVP messages sent
2. The size of the RSVP messages
3. The refresh frequency of path and reservation messages.

The number of RSVP messages is controlled by merging. The size of the messages is in proportion to the number of sources upstream. The refresh frequencies can be tuned; however, the responsiveness to dynamic conditions is dependent on the timing of the refresh frequencies. Currently, static values are used, but in the future, adaptive timeout algorithms will be explored.

Modularity is achieved by minimizing the coupling with the other four components of the architecture: the flowspec, the routing protocol, the network admission control protocol, and the packet scheduling algorithm. The dependencies on the other components are as follows:

- No assumptions are made about the flowspec.

- The admission control protocol provides an admit or reject decision only if all the switches along the path admit the flow.
- The packet scheduling algorithm can change packet filters without needing to establish a new reservation.
- The routing protocol provides both unicast and multicast routing; and a sender to a multicast group can reach all group members.

In the next two sections, the authors review very briefly the operation of the protocol followed by a walk-through of a no-filter reservation and a filtered reservation example, which helps clarify some of their description. Their description relays the following information.

The receiver plays a key role in the operation of the protocol. As the entity responsible for the reservation messages, RSVP must make sure that the reservation message follows *exactly* the data packets of the path message. This involves establishing a sink tree from each receiver to all the sources, which can be accomplished in the following way:

- Combine all the routing trees given by the protocol for the same multicast group, to create a (directed) source mesh; maintain this source mesh by forwarding periodic path messages along the routes provided by the routing protocol.
- Obtain a sink mesh by reversing the direction of all links in this source mesh; then build a sink tree routed at each receiver by tracing the paths from the receiver to reach all the sends.
- Forward reservation refresh messages along the sink trees to maintain current reservation state; however, reservation messages propagate only as far as the closest point on the sink tree where a reservation level greater than or equal to the reservation level being requested has been made.

Each switch uses the path states to maintain, for each multicast group, a table of incoming and outgoing interfaces. Each incoming interface keeps the flowspec information it has forwarded upstream to merge reservation requests from multiple downstream links. For each outgoing link, there is a list of senders, and for each sender the previous hop address for data arriving at the switch, and a set of reservations. A reservation may consist of an owner, a filter, and the amount of resources reserved. The information contained in the reservation depends on the reservation style.

To create and maintain a reservation, each data source sends a path message that contains the flowspec. When a switch receives a path message, it first checks to see if it already has the path state for the named target; if not, path state for that target is created. The switch then obtains the outgoing interface of the path message from the routing protocol and updates its table of incoming and outgoing links accordingly; the source address (plus port number in the case of IP) carried in the path message will be recorded if the path message indicates that the application may require a filtered reservation. The path message is then forwarded only if it is from a new source or a change in route has been detected. Otherwise, the path message is discarded and a new path message containing all path information obtained so far is sent periodically instead.

When a receiver receives a path message from a source for whose data it would like to make a reservation, the receiver sends a reservation message using the flowspec from the incoming message (the flowspec may be modified to meet the needs of the receiver). This message follows the reverse route of the path message to the data source. If any switch rejects the reservation, a RSVP reject message will be sent back to the receiver and the reservation message is discarded.

However, if the reservation message requires a new reservation to be made, it will propagate towards the source until it reaches the closest point along the sink tree where the reservation level is equal to or greater than the one being made.

Once the reservation is established, the receiver periodically sends reservation refresh messages, which are merged as they travel along the sink trees. The details of the merging are dependent on the reservation style and are not elaborated in the paper.

When a sender (receiver) wishes to terminate the connection, the sender (receiver) sends a path (reservation) teardown message to release the path (reserved resources). There is no retransmission timer. If it is lost, the intermediate nodes will time out the corresponding state.

After presenting the basic operation of the protocol, the authors discuss status, and related and future work. In particular, the design has been implemented in a simulator and work is currently underway to implement it on DARTnet. No results from the simulation were included; though, the authors mention that the simulator was used to help revise the design through an iterative test and evaluation process.

The authors then compare their work to ST; ST-II; work by Pasquale, Polyzos, Anderson, and Kompella; and work by Gupta and Moran. The key deficiency of ST and ST-II is that they fail to provide a robust, efficient solution to the multipoint to multipoint case. Furthermore, ST suffers from requiring a centralized access controller to coordinate among the participants and manage the tree. Pasquale et al. have proposed an approach based upon their work with multimedia channels. However, they only considered a single source. Gupta and Moran have proposed a channel grouping approach that is similar to ST in allowing a data source to make a reservation for its own data transmission along its own multicast tree; however, it also allows for each reservation request to specify its relationship to other reservations within its multicast group. Two kinds of relationships have been identified: *advisory* and *mandatory*. However, not enough information was available to make any further comparisons.

The authors conclude with the following suggested list of questions to be explored.

- How does RSVP perform in a real network?
- How well does RSVP scale?
- What other new functionality should be offered (e.g., security and/or authentication)?
- What other reservation styles should be offered?

2.17.2 General Comments

In general, this is a well-written paper. There is some evidence that it is a draft, due to its grammatical errors and incomplete references. Details of the approach are not included; however, the reviewer feels that more information would have been beneficial.

2.17.3 Categorization

Approach for solving resource reservation needs for unicast and multicast applications in a heterogeneous environment.